# MAINTENANCE MEMOS

# PDP-10 TIME-SHARING MONITOR

Memos #1-8
January, 1969

## NOTE

This is a collection of the eight Time-Sharing Internal Memoranda previously issued as separate documents. The chief purpose of publishing them in this form is to make their storage and distribution more convenient. Thus, it should be noted that this collection contains no new or revised information and is not intended to be a complete coverage of the Monitor.

As new and updated information becomes available, it will be published first in the form of new or revised memos to be incorporated in this collection and, eventually, in a formal DEC software manual.

TABLE OF CONTENTS


Memo #1 EXECUTIVE MODE USE OF THE PRIORITY INTERRUPT SYSTEM
The Significance of Priority Levels 1
Priority Channel Assignments 1
Control of the PI System 2
Machine Action upon Interrupt 3
Input/Output Programming 4

Figure 1 Partial Schematic Representation of JSR Exple 5
Figure 2 Diagram of Monitor Priority Interrupt Levels 8

Memo #2 JOB SCHEDULING
Job Scheduling in the 10/40 Nonswapping System 2
CLKCSS 2
Job Scheduling in the 10/50 Swapping System 4
10/50 Scheduling Routines 5
Job Queues and Queue Transfers 8
Swapping 11
Notes on Queue Scanning for Input/Output Swapping 12

Table 1 Some Queue Tables in CSSDAT 4
Figure 1 Schematic of Job Queue Table 8
Figure 2 Job Scheduling, 10/40 Nonswapping System 14
Figure 3 Job Scheduling, 10/50 Swapping System 15

Memo #3 COMMAND DECODER
An Overview 1
Adding a Command 1
Examples 2
Initial Setup and Verification 3
Command Routines 4
START and CSTART Commands 5
R Command 6
RUN Command 6
Cleanup and Return 9

Table 1 COMTAB Bits 2
Table 2 Cross-Reference Listing of Symbols in Command Decoder 20
Figure 1 Command Decoder Flow Chart - Initial Setup and Verification 10
Figure 2 Command Decoder Flow Chart - Command Routines for START, CSTART Commands 14
Figure 3 Command Decoder Flow Chart - Command Routines for R, RUN Commands 15
Figure 4 Command Decoder Flow Chart - Cleanup and Return 18

Memo #4 APR AND CLOCK INTERRUPT ROUTINES
APR Interrupt Routine 2
CLK Interrupt Routine 5
Table 1 Cross Reference Listing of Symbols in APR and CLK Interrupt Routines 13
Figure 1 Relationship between APR and CLK Channels 1
Figure 2 Flow Chart of APR Interrupt Routine 3
Figure 3 Flow Chart of CLK Interrupt Routine 9

Memo #5 PROGRAMMED OPERATOR SERVICE (UUOCON)
Description 1
Operator Preprocessing and Dispatch 2
Special Registers 2
Functional Description 3
Operator Service 4
Exit Routines 6
Error Exits 6
Normal Exits 7
Adding a Programmed Operator 7
Adding a New Operator 7
Adding a New CALL Subfunction 8

Memo #6 SYSTEM INITIALIZATION AND RESTARTS
FIRST 1
SYSINI 3
ONCE 6

Table 1 System Dispatch Table 1
Table 2 System Parameter Values Stored in FIRST 2
Table 3 Priority Interrupt System Trap Locations 6
Table 4 Cross Reference Listing of System Initialization Symbols 13
Figure 1 Map of Upper Section of Monitor 9
Figure 2 System Initialization Flow Chart (SYSINI) 12

Memo #7 CONTEXT SWITCHING
Context Switching 1

Table 1 Cross Reference Listing of Context Switching Symbols 3
Figure 1 Flow Chart of Context Switching 3

Memo #8 I/O OPERATORS AND DEVICE SERVICE ROUTINES
Software Link between User and Device 1
I/O Operators 3
Review of User I/O 3
INIT and OPEN Operators 5
INBUF and OUTBUF Operators 5
INPUT Operator 5
Dump (Unbuffered) Mode 5
Buffered Modes 6
OUTPUT Operator 8
Dump (Unbuffered) Mode 8
Buffered Modes 8
CLOSE Operator 9
RELEASE Operator 10
LOOKUP and ENTER Operators 11
Device Service Routines 24
Device Data Blocks 24
UUO-Level Operations 29
Dispatch Table 29
Basic Operations 29
Interrupt-Level Operations 32
Interrupt Channel Routines CHAN and NULL 32
Interrupt Service 33

| | | |
|---|---|---|
| Table 1 | Device Data Block (DDB) Bit Definitions | 27 |
| Table 2 | Device Service Dispatch Table Entries | 29 |
| Table 3 | Monitor UUO's | 37 |
| Table 4 | CALL SIXBIT/name/ and CALLI n | 39 |
| Table 5 | Cross Reference Table of I/O Programmed Operator Symbols | 41 |
| Figure 1 | JOBJDA or USRJDA Word Contents | 2 |
| Figure 2 | Buffered Data Transfer between an Input Device and User via a 3-Buffer Ring | 4 |
| Figure 3 | Flow Chart of INIT Operator | 12 |
| Figure 4 | Flow Chart of INBUF, OUTBUF Operators | 13 |
| Figure 5 | Flow Chart of INPUT Operator | 14 |
| Figure 6 | Flow Chart of OUTPUT Operator | 17 |
| Figure 7 | Flow Chart of CLOSE Operator | 19 |
| Figure 8 | Flow Chart of RELEASE Operator | 21 |
| Figure 9 | Flow Chart of LOOKUP and ENTER Operators | 23 |
| Figure 10 | Device Data Block (DDB) | 26 |
| Figure 11 | General Flow for Output Interrupt Routine | 34 |
| Figure 12 | General Flow for Input Interrupt Routine | 36 |

# 1

PDP-10 TIME-SHARING MONITORS

# EXECUTIVE MODE USE OF THE PRIORITY INTERRUPT SYSTEM

The PDP-10 incorporates a flexible, seven channel, priority interrupt system that is particularly useful in programming efficient multiple input/output operations. The purpose of this section is to acquaint the user with some of the programming techniques involved with using this system. Topics to be discussed include

1. The manner in which input/output (I/O) devices are connected to the interrupt channels;
2. Programmed control of the PI system;
3. The action taken by the system upon acknowledgement of an interrupt request from a device;
4. The action most appropriately taken by the programmer in anticipation of such an interrupt.

## 1. THE SIGNIFICANCE OF PRIORITY LEVELS

The seven priority interrupt channels are numbered according to their priority level, with channel 1 having the highest priority. When an interrupt request on a given channel is being serviced, no further interrupts can occur on that channel or on any channel of a lower priority; however, a channel of a higher priority can interrupt the routine servicing the original interrupt. The system is designed so that the original routine is resumed following the servicing of the higher priority interrupt. Further, requests occurring on lower priority channels are never lost, but are simply held until such time as they can be acknowledged and serviced. In general, an interrupt request from a device consists of a level present on one of the seven PI request channels (lines) which are part of the I/O bus. This level remains present until the device is serviced.

## 2. PRIORITY CHANNEL ASSIGNMENTS

Up to 126 input/output devices can be connected to the central processor via the I/O bus. Under control of the Monitor, one or more devices can be connected to any one of the seven priority interrupt channels. A particular device is connected to a channel with a Conditions Out (CONO) instruction directed to that device. The CONO instruction contains the channel number in its effective address portion, usually in bits 33 through 35.

```
Example     CONO PTR, 000005      ;ASSIGNS THE PAPER TAPE READER TO
                                  ;CHANNEL 5, SO THAT WHENEVER THE
                                  ;READER'S "DONE FLAG" IS SET, AN
                                  ;INTERRUPT WILL BE REQUESTED ON
                                  ;CHANNEL 5
```

Note that in this example, as in all others to follow, the coding is presented in a format acceptable to the Macro-10 Assembler and that all numbers are in octal. After this instruction has been executed, the paper tape reader will request an interrupt each time its "done flag" is set; whether or not the request is acknowledged depends on the state of the PI system, a condition entirely within control of Monitor (this is discussed below).

## 3. CONTROL OF THE PI SYSTEM

The PI system itself is considered to be an I/O device and is controlled by a Conditions Out (CONO) instruction with a device code of 004. As in the case of other I/O devices, the PI system may be thought to contain a control register whose bits are set according to the bits in the effective address of the CONO instruction. The significance of these control bits is summarized below. Note that in this summary the term "selected channels" refers to those channels corresponding to 1's in bits 29 through 35 of the control register (the effective address of the CONO instruction), where bit 29 corresponds to channel 1, bit 30 to channel 2, etc.

| BIT | OCTAL | FUNCTION |
|---|---|---|
| 23 | 10000 | Clear the entire PI system. |
| 24 | 4000 | Activate an interrupt on the selected channels. |
| 25 | 2000 | Turn on the selected channels. |
| 26 | 1000 | Turn off the selected channels. |
| 27 | 400 | Turn on the PI system. |
| 28 | 200 | Turn off the PI system. |

Bit 23, if a 1, cancels all previous requests, turns off all channels, and turns off the PI system. Bit 24 is used to request an interrupt on a different priority channel than the one which is active. Bits 25 and 26 allow the user to turn on or off (but not both in any single instruction) any desired channel or channels. A request level present on the I/O bus line connected to a channel which has been turned off will not be acknowledged, but the level remains present awaiting the reactivation of the channel. Thus, the user can delay an interrupt or prevent it from occuring at an inopportune time by turning off the appropriate channel.

The entire PI system can be turned off with bit 28. From the viewpoint of external devices, the system appears as if it were permanently servicing a request on a channel with a higher priority than channel 1. Any interrupt requests which occur will be acknowledged if their respective channels are on, but they will not be serviced until the system is turned back on with bit 27.

Some examples may serve to illustrate these concepts.

```
CONO PI, 10000          ;CLEAR THE PI SYSTEM.  THIS
                        ;INSTRUCTION MAY BE USED TO ADVANTAGE
                        ;IN THE INITIALIZATION SECTION OF A
                        ;PROGRAM USING THE PI SYSTEM.

CONO PI, 1007           ;TURNS OFF PI CHANNELS 5 THROUGH 7.

CONO PI, 12577          ;CLEAR THE PI SYSTEM, TURN ON THE PI
                        ;SYSTEM, AND TURN ON ALL SEVEN
                        ;CHANNELS.
```

Note that conflicting requests (e.g., both bits 27 and 28 set to 1) will yield unpredictable results; the "clear PI system" operation (bit 23), however, does not conflict with any other operation and occurs first when the CONO instruction is executed. Also, the following two instructions are equivalent in effect, if channel 1 is the only channel being used.

```
CONO PI, 200            ;BIT 28 - TURN OFF THE PI SYSTEM

CONO PI, 1100           ;BITS 26 AND 29 -  TURN OFF CHANNEL 1
```

## 4. MACHINE ACTION UPON INTERRUPT

When an interrupt level appears and the selected channel is free and no higher priority channel is in use, an interrupt is granted at the end of the instruction in progress. The mechanism is as follows.

Control is transferred to core memory location 40 + 2n, where n is the channel number. The program counter is not affected in any way by the interrupt unless the instruction in location 40 + 2n changes the program counter during execution (if, for example, this location contains a jump-type instruction, it is the programmer's responsibility to preserve the contents of the program counter if he has any intention of returning to the interrupted program sequence). The system is designed so that the instruction in location 40 + 2n should be one of the following:

```
JSR
BLKI
BLKO
```

Each of these will be considered in detail later. While other instructions are not illegal, their use is never necessary and should, in general, be avoided.

One further point should be understood: when an interrupt is serviced (when the program sequence beginning in location 40 + 2n is being executed), the PI system is disabled to the extent that further interrupts may not occur on the channel currently in use or on any lower priority channel. This condition prevails until the program dismisses the channel in use. This action of dismissing the channel must be taken by the program before control is returned to the interrupted sequence if any further use of the affected channels is expected. There are only two ways in which a channel can be dismissed: the first is

through the execution of a JRST instruction with bit 9 equal to 1; the other is through the execution of a BLKI or BLKO instruction, both of which will automatically dismiss the current channel if the transfer of a data block is still incomplete. These concepts will be illustrated and clarified in the programming examples which follow.

## 5. INPUT/OUTPUT PROGRAMMING

NOTE

It is assumed that the reader is familiar with the operation of the JSR and JRST instructions as well as the eight I/O instructions given in the PDP-10 System Reference Manual.

Consider first the use of a JSR instruction in location 40 + 2n. As a specific example, consider the instruction

```
JSR 1000
```

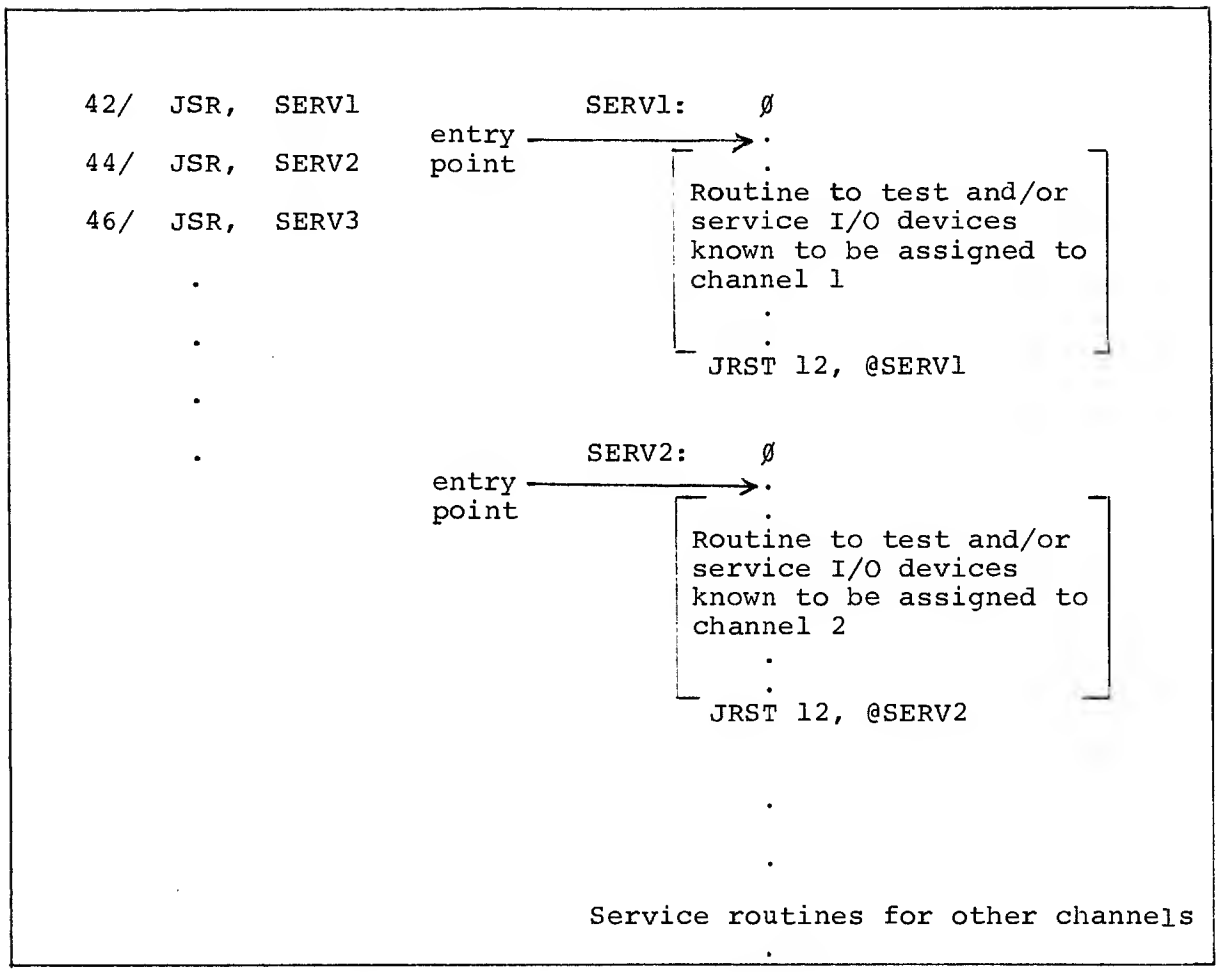
in location 44, to which control is transferred when an interrupt request is serviced on channel 2. The state of the flags and the program counter (which is pointing to the instruction which was about to be executed when the interrupt occurred) is stored in location 1000; control is then transferred to location 1001, with channels 2 through 7 disabled. Beginning at location 1001 should be a routine to service the device connected to channel 2. If several devices are connected to channel 2, the routine must contain appropriate CONSI or CONSO instructions to determine which "done flag" has been set. The last instruction in the routine should be a JRST 12, @1000. The specification of AC 12 causes bits 9 and 11 to be 1's; bit 11 specifies that the flags stored in location 1000 are to be restored to their former states and bit 9 causes the PI channel currently in use (channel 2) to be dismissed, thus freeing channels 2 through 7. Control is then transferred to the location specified in the address portion of location 1000 (the interrupted sequence). The execution of the routine beginning in location 1001 might have been interrupted by a request from a device assigned to channel 1. If, in location 42, there was a JSR to a similar service routine which ended with its own JRST 12, @nnnn, then control would automatically transfer back to the channel 2 routine and from there to the original interrupted sequence.

The above techniques may be extended to cover all seven channels and are sufficient for full utilization of the PI system. A partial schematic representation of the program structure appears on the next page.

When blocks of data must be transmitted into or out of core memory, especially when it is desired that the transfer take place at the maximum rate the I/O device allows, the BLKI and BLKO instructions may be used to advantage with the PI system. The technique is somewhat different from that of the JSR example above. As a specific example, consider the case of reading three words from paper tape into memory locations 6000 through 6002 while performing some computation elsewhere. The instruction

```
CONO PTR, 63
```

assigns the paper tape reader to channel 3 and causes one word (six




Partial Schematic Representation of JSR Example

frames) of tape. The instruction

```
CONO PI, 12420
```

clears the PI system, turns it on, and turns on channel 3. When the reader "done flag" becomes a 1, an interrupt is requested on channel 3 and is serviced at the completion of the instruction in progress. Control is transferred to location 46, which should contain the instruction

```
BLKI  PTR, BPWD
```

where the block transfer pointer word is defined elsewhere using the IOWD pseudo-instruction

```
BPWD: IOWD  3, 6000
```

The execution of the BLKI instruction proceeds in the usual (non-PI) manner, except that when the single word transfer is complete and the pointer word has been tested for an end-of-block condition, one of two actions is taken by the hardware.

a. If the last data word of the block has <u>not</u> been read in, the interrupt channel currently in use is <u>automatically</u> dismissed and control is returned to the interrupted sequence (pointed to by the program counter).

b. If the last data word has been read in, the channel is <u>not</u>

dismissed, and control goes to the instruction following the BLKI instruction (in this case, location 47). The program counter is still pointing to the interrupted sequence, but it can be lost at this point through careless programming. The safest instructions to have in location 40 + 2n + 1 are JSR instructions to dismissal routines. In this example, the instruction

```
JSR  DISM
```

in location 47 might be used to complete the input operation by jumping to the brief routine

```
DISM:  Ø        ;BLANK REGISTER FOR PC
                ;AND FLAGS

       JRST 12, @DISM
```

which would dismiss the channel and return to the interrupted sequence. Alternately, the routine beginning at DISM might turn off the PI system or take any other desired action before returning. There are slight differences between input operations and output operations. The reader should refer to the System Reference Manual for these distinctions.

Here is another example, this one of the output variety. The user desires to punch out a block of $100_8$ locations, beginning at LIST, in binary format on paper tape while an independent program is running. Assume that the main program has executed the following three instructions to initiate the process.

```
MOVE  17, IOWD 77, LIST + 1     ;IOWD XWD - 77, LIST

CONO  PTP, 41

DATAO PTP, LIST
```

The first of these three instructions sets up a pointer and counter word in AC 17. The next instruction sets the punch to binary mode and assigns it to PI channel 1. The last of these three instructions activates the punch and punches the first word. When the punch has finished punching the contents of LIST, its "done flag" is set and an interrupt occurs on channel 1. Consider the following two program sequences to service the interrupt. Assume that it is desired to turn off channel 1 to prevent further interrupts until after the last data word has been punched. The two sequences accomplish the same task. Note the manner in which each extracts the second data word correctly from LIST+1.

SEQUENCE 1

```
42/ JSR OUTPUT


OUTPUT: Ø
        DATAO PTP,1(17)
        AOBJN 17, .+2
        CONO  PI, 1100
        JRST  12, @OUTPUT
```

SEQUENCE 2

```
42/ BLKO PTP, 17
43/ JSR  FINISH


FINISH: Ø
        CONO PI, 1100
        JRST 12, @FINISH
```

One final point: the use of the arithmetic processor as an I/O device. The processor can be assigned to any PI channel by a CONO instruction having a device code of Ø (mnemonic = APR). With the arithmetic processor so assigned, an interrupt is requested on the assigned channel whenever any one of the six flags listed below is set.

Memory Protection flag[1]

Nonexistent Memory flag[1]

Clock Count flag (if enabled)[1]

Floating Overflow flag (if enabled)

Overflow flag (if enabled)

Pushdown List Overflow flag (if enabled)[1]

A program designed to service an interrupt requested by the arithmetic processor would have to contain a series of Condition Skip instructions to determine which of the above flags caused the interrupt.

The diagram on the following page illustrates the priority interrupt levels, their functional relationships, and their relative processing intervals.

[1] The Time-Sharing Monitors always enable these flags for all users. The others, too, can be enabled privately by request of a user program.

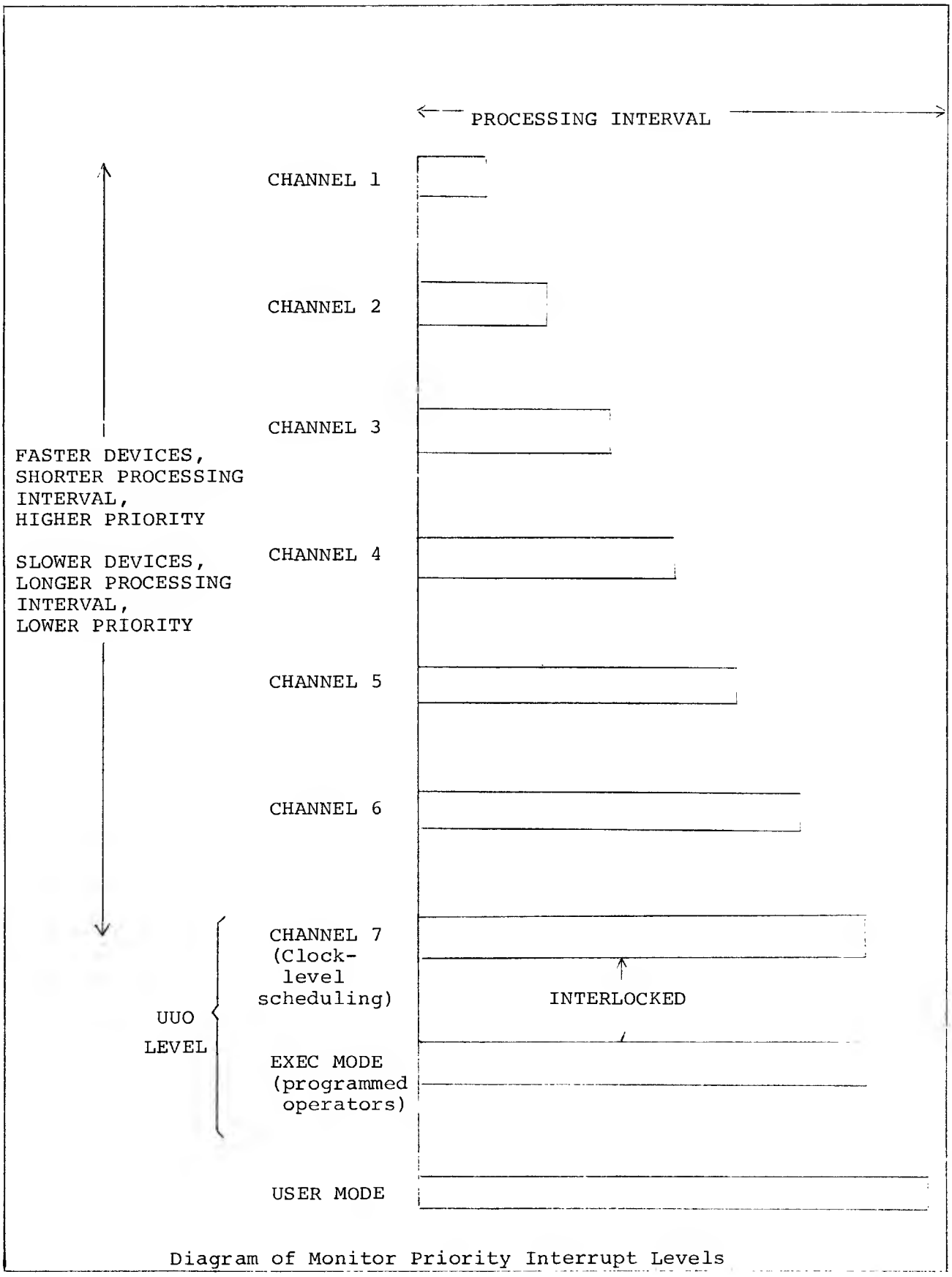



Diagram of Monitor Priority Interrupt Levels

# 2

# JOB SCHEDULING

The following two sections describe the scheduling and queueing of jobs in both the PDP-10/40 and PDP-10/50 Monitor systems.

In the 10/40 system, all jobs reside in core and the scheduler decides which of these jobs should run. In the 10/50 system, jobs can exist on an external storage device (usually disk) as well as in core; the scheduler not only must decide which job to run but also when a job is to be swapped out to the disk or brought back into core.

Jobs are retained in queues of varying priorities which reflect the condition of the job at any given moment. For example,

Run queues - For runnable jobs waiting to execute.

I/O Wait queues - For jobs waiting while doing I/O.

I/O Wait Satisfied queues - For jobs waiting to continue running after finishing I/O.

Sharable Device Wait queues - For jobs waiting to use a sharable device.

Teletype Wait Satisfied queues - For jobs having completed a Teletype operation and awaiting action.

Each of these queues is addressed through tables such that the position of a queue's address in a table represents the priority of that queue with respect to the others. Within each queue, the position of a job determines its priority with respect to the other jobs in the same queue. The queues in the 10/50 swapping system are, of course, more complex than in the 10/40 system.

Scheduling occurs at each clock tick and may also be "forced", that is, it may be called at Exec (Monitor) level between clock ticks if the current job becomes unrunnable.

The sequence of scheduling routines is as follows.

<u>10/40 Nonswapping System</u>

APRSER - Receives interrupt.

NXTJOB/CLKCSS - Performs simplified requeueing, calls shuffler (if required), and schedules which job to run.

APRSER - Dismisses the interrupt.

10/50 Swapping System

APRSER - Receives interrupt.

NXTJOB/SCHEDU - Determines which jobs require requeueing and which procedures are to be used.

QXFER - Performs transfers of jobs among queues.

SWAP - Makes swapping decisions, calls shuffler when required, and initiates and completes swapping I/O.

SCHED - Selects highest priority job to run.

APRSER - Dismisses interrupt.

The routine APRSER is the same in both systems; the others are not the same. The routines to be included in a particular Monitor are determined during the Build process.

## I. JOB SCHEDULING IN THE 10/40 NONSWAPPING SYSTEM

The routine CLKCSS selects and assigns a job to be run for the duration of the next clock tick. The job is selected by scanning the wait and run queues.

CLKCSS is called

a. At each clock interrupt, and

b. When the current job becomes unrunnable.

A general flow diagram of CLKCSS can be found at the end of this section.

### CLKCSS

The following is a narrative description of the CLKCSS routine.

NXTJOB
XCKCSS — Core shuffling is performed, if required by the routine CHKSHF. If this call to CLKCSS was from a clock interrupt, the quantum run time of the current job is decremented (unless the current job is the null job).

NXTØ — If the quantum run time has become zero, it is restored to the time allotted for a running job. An internal flag (RNAVAL) is set, which will cause the run queue to be searched before this job is run again. This flag is also set if the null job is running. The run queue contains all runnable jobs that are not waiting for I/O.

NXT2, NXT3 The I/O Wait queue is searched by assigned priority (fastest device to slowest device). A nonzero position indicates that some job is waiting at that particular queue level. The Run queue, which has the lowest priority, is searched only if RNAVAL is set and no job is waiting at a higher priority level. If all queue positions are zero, control is transferred to NXT7 (this is the only entry to NXT7).

NXT5 At this point, a nonzero queue has been found, indicating that a job is waiting at that particular queue level; this nonzero value is also the index register value. All job numbers except Ø (the null job) are now searched for three conditions

1) Runnable (bit Ø, job status word);

2) Job number is assigned and job is initialized (bit 3, job status word); and

3) Job is waiting at this I/O queue level (wait code in job status word, bits 10 through 15, matches index value of I/O Wait queue).

Since it is possible for more than one job to be waiting for the same device, a search is made giving the lowest priority to the last job to use the device in a round-robin fashion. (Note that a specific device may not be implied since the queue table includes jobs in an I/O Wait Satisfied condition and jobs in the Run queue.) It is also possible for a job to have become unrunnable after having been placed in the I/O Wait queue, and if no waiting job is found at the indicated level, the search is continued at the next lower level by returning to NXT3.

NXT8 A waiting job has been found for the indicated queue and the queue is cleared. (Two positions in the queue, I/O Wait Satisfied and TTY Wait Satisfied, are enabled for multiple jobs. In these cases only, the queue contains a count instead of a flag. This count is decremented.) The number of the job to be run is saved in a table, JOBP, and the job's status word is set for the proper run time and wait code. NXT8 exits with the job number in the accumulator, ITEM.

By saving the number of the job to be run in the table, JOBP, this routine gives itself the ability to assign the lowest priority for any device to the last user of that device.

NXT7 This portion of the routine is reached only when the I/O Wait queue is empty. If the flag, RNAVAL, is set, the Run queue has also been searched and no runnable job was found. RNAVAL is then cleared, ITEM (containing the number of the job to be run) is set to Ø, and this routine exits to the null job.

If the flag, RNAVAL, is not set, the Run queue has not been searched. The flag is now set and a branch is made to NXT5 with the job number in ITEM set to the currently running job. This effects the Run queue search by giving the highest priority to the job which had been running.

Table 1
Some Queue Tables in CSSDAT

| Index Value | Position | Use |
|---|---|---|
| 0 | RNAVAL | If set to nonzero, search all jobs for one in runnable state. Start search with current user if his quantum time is greater than Ø. |
| 1 | WSAVAL | Contains a count of jobs in I/O Wait Satisfied condition. Count is decremented whenever a job is run from this queue level. |
| 2 | TSAVAL | Contains a count of jobs in TTY Wait Satisfied condition. Count is decremented as in WSAVAL. |
| 3<br>4<br>.<br>(maximum) | xxAVAL<br>"<br><br>" | Queue positions of I/O devices in increasing order of priority. If Ø, no jobs are waiting for device; if nonzero, some job is waiting. Set to Ø when a job is run from that position. |
| 0<br>1<br>.<br>.<br>(maximum) | JOBP | Each position saves job number of last job that was run from the corresponding position in the above queue. Enables CLKCSS to share job priority from each queue level. |

## II. JOB SCHEDULING IN THE 10/50 SWAPPING SYSTEM

The following section describes the activities involved in scheduling, rescheduling, an requeueing jobs in the PDP-10/50 system.

Each job number possible in the system resides, at any moment, in one particular queue (job numbers not assigned reside in the Null

queue). There are three Run queues of different levels of priority from which jobs are run, and each job when run is assigned a quantum time. When this quantum time expires, the job will cease to run and will be moved to a lower priority Run queue. If for any reason all the jobs in the first two Run queues cannot be run, one is selected from the third Run queue; when its quantum time has expired (been decremented to Ø), it is moved to the second Run queue - thus, users in queues two and three are run in a round-robin manner.

The activities of a job currently running may cause it to enter into one of several special states

Waiting to do I/O
Waiting to access a sharable device
I/O complete (satisfied)

All special states need not be dependent on I/O conditions, as in the case of the program operator SLEEP.

Each special state has a queue of jobs which are in that particular state.

Example A currently running job begins input from a DECtape, the job is placed into a wait queue, the input is begun, and a second job is set to run while the first job waits and its input proceeds. The second job then decides to also access the DECtape control for an I/O operation. Since the DECtape control is busy, the second job is stopped, put into a queue for jobs waiting to access the DECtape control, and a third job is set to run.

The input operation of the first job then finishes, making the DECtape control available. The second job is found to be waiting for the control. The second job's I/O operation is initiated and the second job is then transferred from the Device Wait queue to the I/O Wait queue. The first job is transferred from the I/O Wait queue to the highest position of the three Run queues. This permits the first job to now pre-empt the third job and run. When its quantum time becomes Ø, it will be moved back to the second Run queue and the third job runs again until the second job completes its I/O.

## 10/50 SCHEDULING ROUTINES

These routines are executed every clock tick to see if a job with a higher priority than the one currently running is waiting.

If the currently running job changes its state so that it becomes unrunnable, these routines are executed immediately from Monitor level without waiting for a clock interrupt.

### CLKCSW

NXTJOB If the call was a result of a clock tick, the "in core protect" time of each job in core is decremented, as is the quantum time of the running job.

If the call was a result of the current job becoming unrunnable, the current job is requeued by a branch to CKJB3; otherwise,

control proceeds to CKJB1 (however, if the current job's quantum time has decremented to Ø, the subroutine QXFER is executed to requeue the job before going to CKJB1; the options specified to QXFER are to reset the job's quantum time and to place the job in a new queue as a function of its present queue).

CKJB1 Location QJOB has a bit set for each job in need of requeueing. The bit position determines the job number. If no requeueing is requested, control goes to CKJB5; otherwise, the job number of the first job in need of requeueing is obtained.

CKJB3 The wait code is retrieved from the job status word of the job. Before the queue transfer routine, QXFER, is called, the method of transfer must first be determined. This is accomplished by specifying the entry to the queueing transfer table used by QXFER and controls the destination queue of the transfer and the quantum time reset conditions. At this point, three different conditions are recognized and handled as follows.

1) The job is in command wait condition and is on the disk. Set transfer table conditions to select the Command Wait queue as the destination queue.

2) The job is not runnable. Set the transfer table conditions to select the Stop queue.

3) The job is runnable. Set the transfer table conditions to select the destination queue and the quantum time reset conditions according to the job's wait code. Since the job is being requeued, its wait code must have changed recently.

QXFER is then executed and requeueing is accomplished. If this requeueing is the result of the current job becoming unrunnable (a forced clock call), a nonreturning call is made to SCHED to select the next job to run. If this is an actual clock interrupt call and more jobs need to be requeued, a jump is made back to CKJB1. When all requeueing requests have been serviced, control proceeds to CKJB5.

CKJB5, CKJB6 The list of all sharable devices is scanned to determine if any are available; this list also contains the count of the jobs in TTY Wait Satisfied and in I/O Wait Satisfied. The flags in this list are set in APRSER. If no sharable devices are available, control proceeds to CKJB7; otherwise, control goes to CKJB6A.

CKJB6A — If a device has become available, its queue is searched for a job which is waiting for the device and is in core. If such a job is found, it is requeued. The address of the queueing transfer table entry for QXFER is determined by a correspondence table (a table specifying transfer table entries for QXFER for each position of the sharable device available table). The sharable device table flag is cleared (the count is decremented for TTY Wait Satisfied and I/O Wait Satisfied), and, except for the last two cases where requeueing is not necessary, the job is requeued by calling QXFER.

CKJB8 — If all positions in the sharable device available table have not been scanned, the routine loops back to CKJB6 to assure a complete scan.

CKJB7 — At this point, all requeueing has been performed and the SWAP routine is called.

SWAP — SWAP handles all input/output of jobs between core memory and the disk. When called, it completes any I/O started earlier, searches for a job to bring in, and (if necessary) finds a job of lower priority to output in order to bring in the new job. When finished, SWAP calls SCHED.

SCHED — SCHED scans the queues of runnable jobs forward (highest priority first). When a job is found, it must be determined that the job is currently in core. The queue location of the job is retained for later requeueing, the wait code in the job status word is cleared, and a return is made to APRSER with the job number in ITEM, causing the job to run. If no job is found by the queue scan, the null job is run.

NOTE

A flow chart of these scheduling routines can be found at the end of this section.

## JOB QUEUES AND QUEUE TRANSFERS

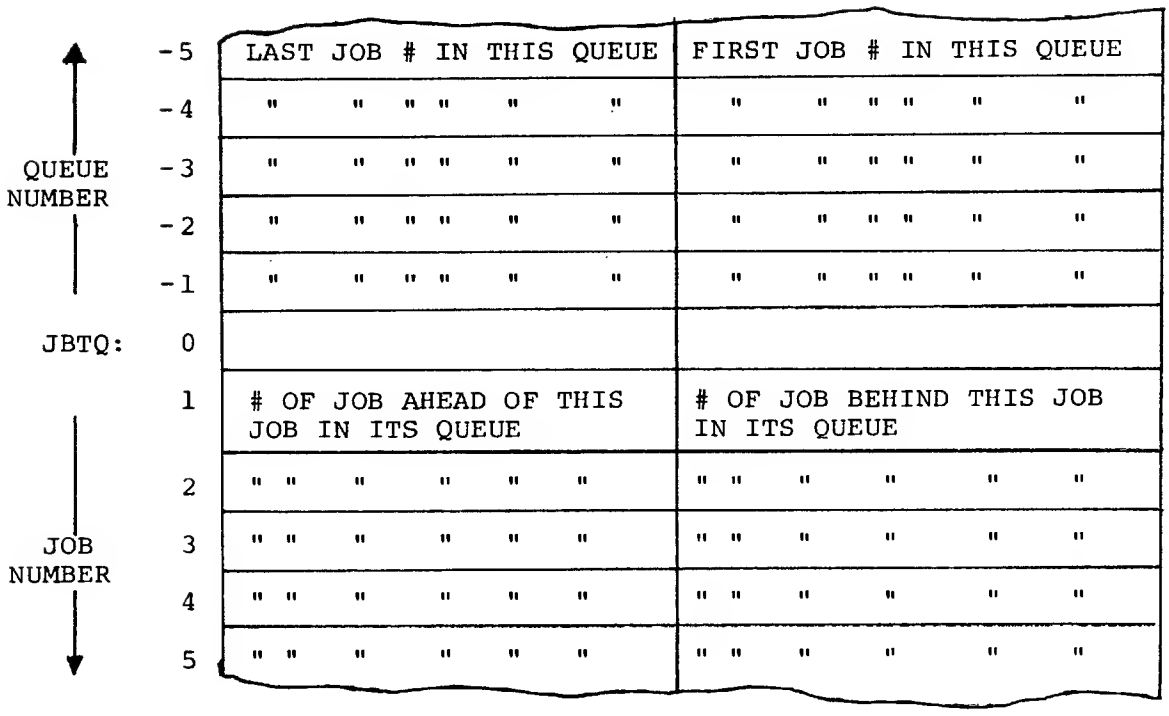



Figure 1. Schematic of Job Queue Table

There is an entry (queue word) for each queue and an entry (job word) for each job. Each queue word specifies the first and last job the queue contains. Each job word specifies which job precedes the particular job and which job follows it in the queue. If a job is the first one in its queue, the left half of its job word contains the number of the queue (negated); if a job is the last one in its queue, the right half of its job word contains the number of the queue (negated).

To determine which jobs are in queue n:

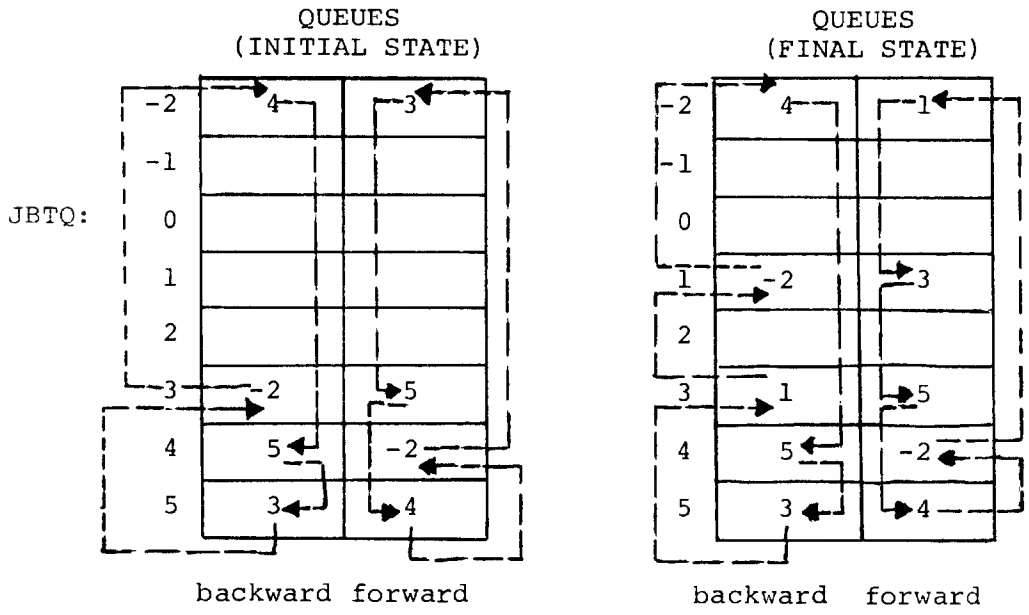
1) Retrieve the numbers of the first and last jobs in the queue from the queue's queue word.

2) Retrieve the job numbers from the job word of the first job in the queue. The left half of the job word should contain the queue number (negated). If the job number in the right half is not the last job in the queue, retrieve its job word.

3) Check that the left half of this job word contains the job number of the above job. If the job number in the right half is not the last job in the queue, retrieve its job word. Repeat this process until the last job is found.

Thus, the numbers of the jobs and their positions in the queue are defined. This procedure may be reversed so that a scan can be performed backwards (from last job to first job) as well as forwards through any queue, and a scan may not only go completely through a queue but may stop on the first or last job in the queue while scanning in either direction. All of these methods are employed in the queue searches performed.

Also, since both the beginning and the ending of a queue are marked, a job can be inserted at either end by merely modifying the linkages. An example is given below by illustrating the method used in inserting JOB1 at the beginning of QUEUE 2.




The above process can be logically described as follows.

$$TEMP_R \leftarrow Q2_R$$

$$Q2_R \leftarrow 1$$

$$JOB1 \leftarrow -2, TEMP_R$$

$$JOB(TEMP)_L \leftarrow 1$$

QUEUE TRANSFERS

When a job is transferred from one queue to another, the destination queue to which it is transferred may be determined by one of three methods.

1) By the job's size

2) By the queue from this the job is being transferred (the source queue)

3) By a change in the job state (fixed destination)

A job in the Run Wait queue being transferred to one of the three Run queues is assigned on the basis of its size, the larger jobs going to the lower priority queue.

A job being transferred from one Run queue to another when its quantum time is exhausted is assigned on the basis of the source queue.

A job waiting to access a sharable device is assigned to the Wait queue for that specific device.

When a job is inserted into a queue, it may be placed either at the beginning or the end of the queue. A job attempting to access a sharable device is placed at the end of the device's Wait queue. When the device becomes available to a job which has been waiting, the job is placed at the beginning of the first Run queue where it pre-empts other jobs and runs immediately, resulting in maximum utilization of the device.

When the queue transfer routine, QXFER, is called, an accumulator contains an address in a transfer table. From this address, the routine accesses two words which specify the following conditions.

How the destination queue is to be determined;

At which end of the queue the job is to be inserted; and

If the quantum time is to be reset.

The transfer table is accessed by using the wait code of the job status word as an index value through a table of pointers. The transfer table entries for each of the three types of queue transfers (fixed destination, destination queue determined by source queue, and destination queue determined by job size) are given below.

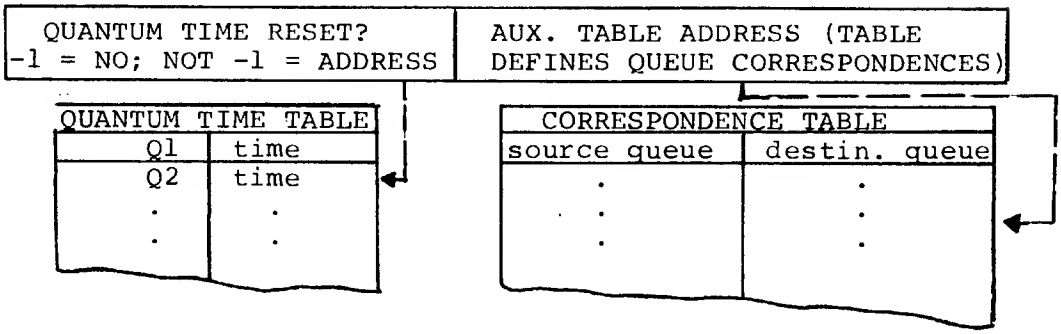
1. Fixed Destination

| Ø = BEG. OF QUEUE<br>1 = END OF QUEUE | FIX |
|---|---|
| QUANTUM TIME RESET?<br>-1 = NO; NOT -1 = YES | NUMBER OF<br>DESTINATION QUEUE |

2. Destination Queue Determined by Source Queue

| Ø = BEG. OF QUEUE<br>1 = END OF QUEUE | LINK |
|---|---|

2. Destination Queue Determined by Source Queue (Cont.)




3. Destination Queue Determined by Job Size

| 0 = BEG. OF QUEUE<br>1 = END OF QUEUE | JSIZ |
|---|---|

| QUANTUM TIME RESET?<br>(SAME AS #2 ABOVE) | AUX. TABLE ADDRESS (TABLE DEF. CORRESP. BETW. SIZE AND QUEUES) |
|---|---|

## SWAPPING

The swapping routine, SWAP, is entered at each clock tick and has the inherent task of bringing a job from disk into core. This function is dependent upon the core shuffling routine, which consolidates "holes" (unused areas) in core so as to make sufficient room for the incoming job, and upon the swapper, which (if required) creates additional room in core by transferring jobs from core to the disk.

Since a considerable amount of I/O may be involved in both shuffling and swapping, either operation may continue over more than one clock tick. The first duty of the swapper is, therefore, to test for conditions which could exist from a previous action performed by the swapper itself directly or indirectly. These conditions are

1. Busy doing I/O from last swap;

2. Waiting for the shuffler (which may be waiting for a job to become shuffable);

3. I/O transfer completed since last clock tick, but error checking and bookkeeping duties remain; and

4. Still trying to fit a job in or force a job out.

When the shuffler is entered, if either the shuffler is waiting or a previous swapping I/O operation is still in process, the routine cannot continue at this time and exits.

If swapping I/O has been completed since the last clock tick, I/O errors are checked. If an output error occurred, the output operation is retried; if the output was completed without error, the job's core is reclaimed and the routine continues at SWP1. If an input error occurred, a message is printed on the console Teletype and an exit is made to APRSER to force rescheduling; if input was completed without error, the disk area is reclaimed, relocation and protection registers are stored along with the "in core protect" time, and the job's swap flag is cleared (the job's swap flag is cleared only while the job is in core; it remains set while the job is on disk or in transit).

SWP1 Since the I/O bookkeeping is complete, the flag (FINISH) which indicates this is cleared. The flags FORCE and FIT are then tested to determine if, on a previous clock tick, the routine was attempting to fit a job into core (if so, go to FIT1) or force a job out of core to the disk (if so, go to FORCE1).

If neither condition exists, the job queues are scanned forward (highest priority first) for a job to bring into core. If none is found, a check for jobs trying to expand their core allocation is made; if none are found, the swapper has nothing more to do and exits.

If the queue scan does locate a job on the disk to input (or if the FIT flag was set - see above), control proceeds to FIT1.

FIT1 The job size is checked against the largest unused area in core and input is begun if the job will fit. If the job will fit only by shuffling jobs, the FIT flag is set and shuffler is called. If it is necessary to output a job, the first jobs checked for are those trying to expand; then the queues are scanned backwards for a job of lower priority than the job being brought in and whose "in core protect" time has been exceeded. It may be necessary to scan for more than one such job to obtain enough core for the incoming job. If sufficient core cannot be obtained, the routine exits.

The largest of the jobs found during this search is chosen for swapping out and the FORCE flag is set. Processing continues at FORCE1.

FORCE1 (This routine may also be reached from the FORCE flag test at the beginning of swapper.)

If the job selected for swapping out has active I/O devices, or is the current job (whose protected area may not yet be restored to the job data area), the output is delayed and the routine exits.

## NOTES ON QUEUE SCANNING FOR INPUT/OUTPUT SWAPPING

When the job queues are scanned for a job to input (scan is forward) or a job to output (scan is backward), a slight deviation is made for the

queues of sharable devices. When scanning for input, only the first job found in each queue is selected for input; when scanning for output, all jobs except the first are selected for output. This process helps to ensure that each device will have a job waiting when the device becomes available.

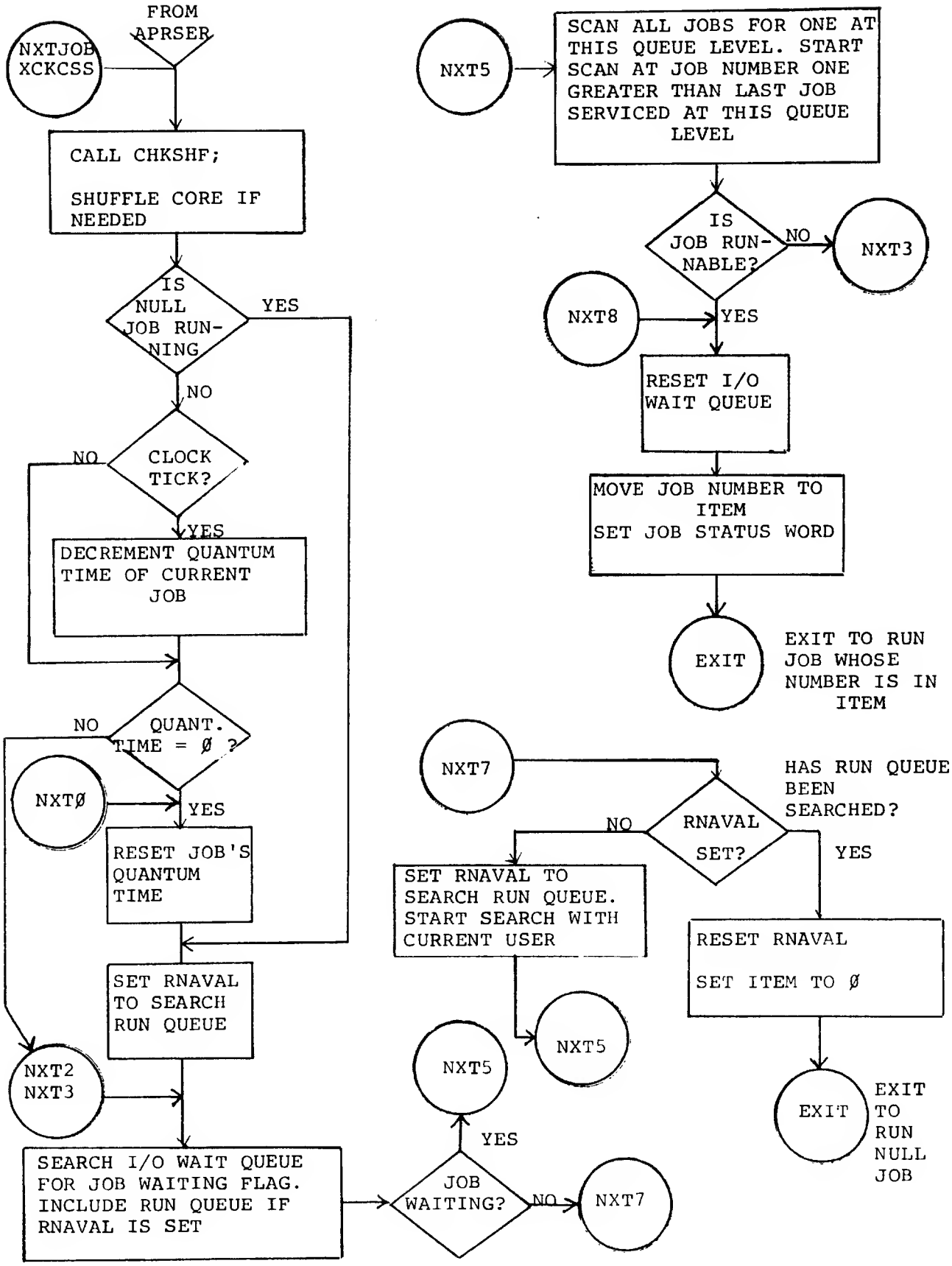



Figure 2. Job Scheduling, 10/40 Nonswapping System

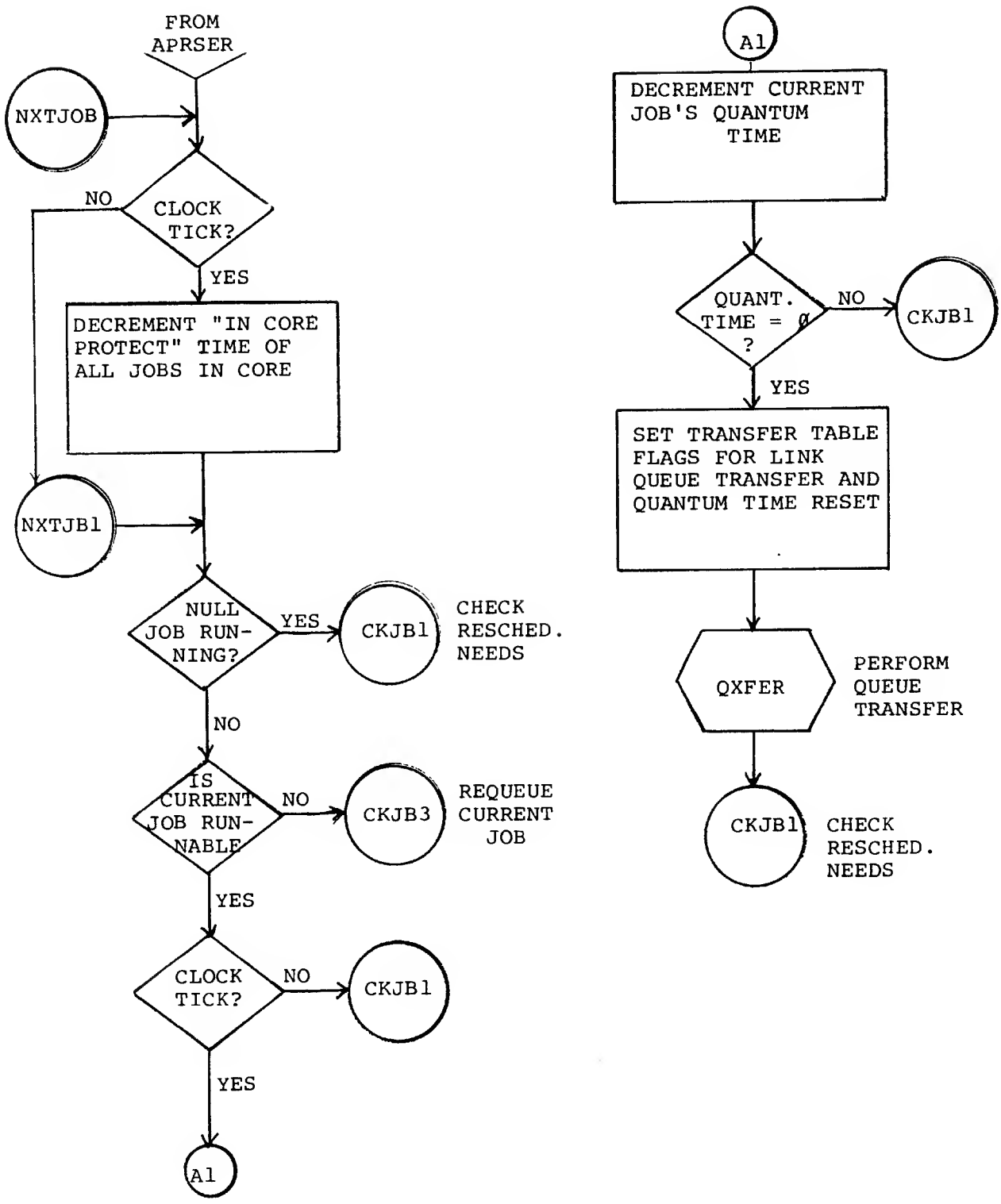



Figure 3. Job Scheduling, 10/50 Swapping System

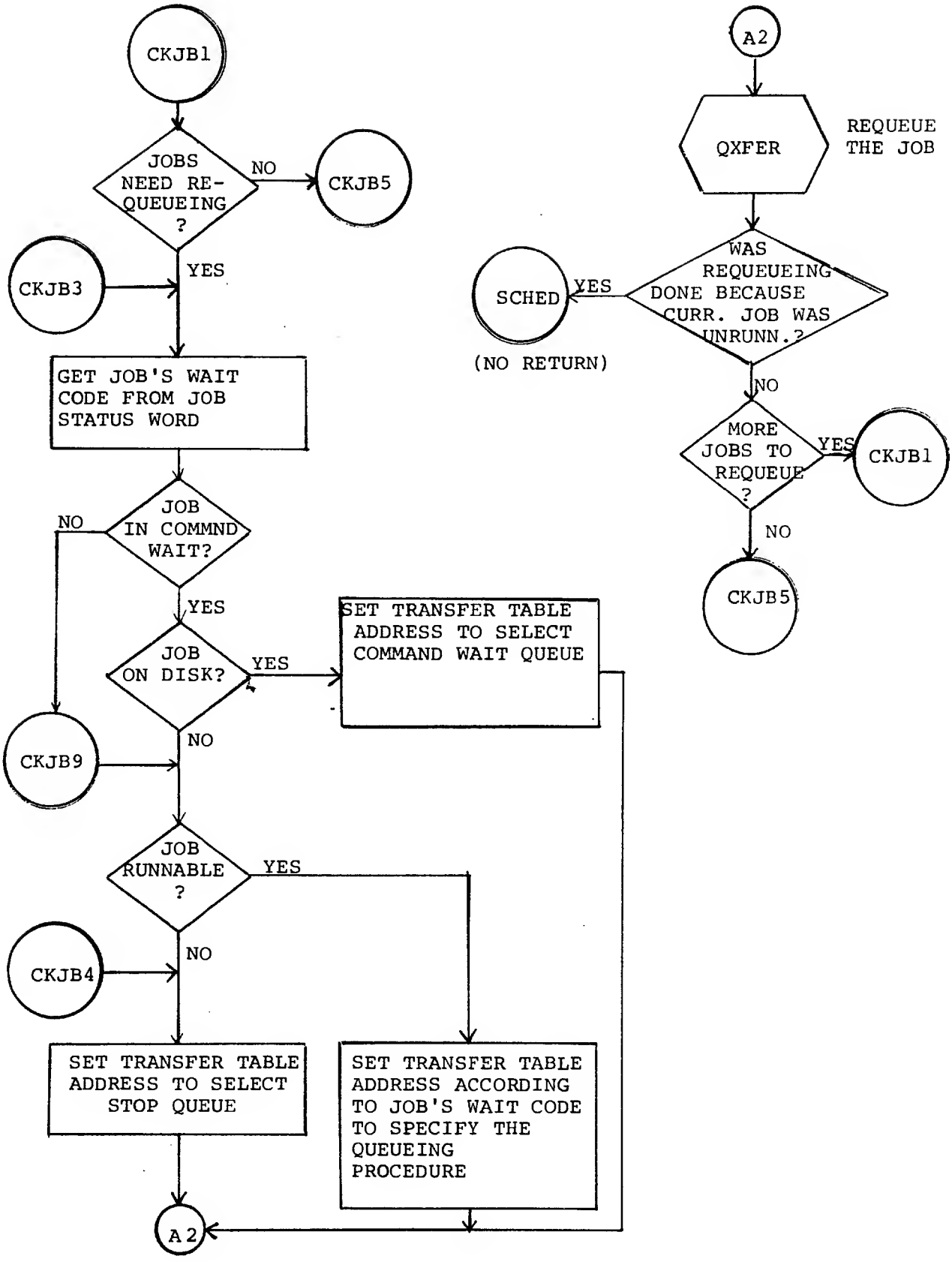



Figure 3 (Cont.) Job Scheduling, 10/50 Swapping System

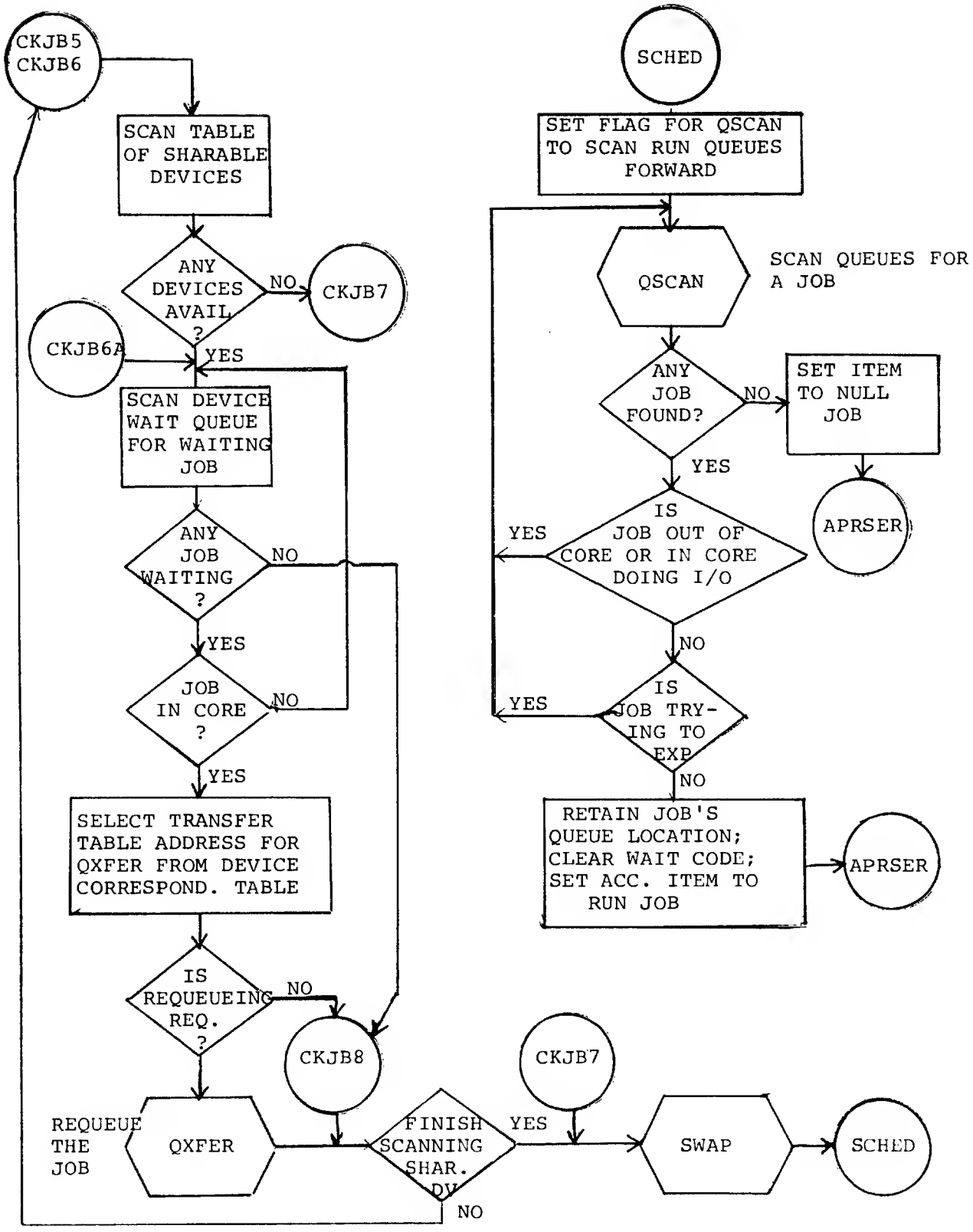



Figure 3 (Cont.) Job Scheduling, 10/50 Swapping System

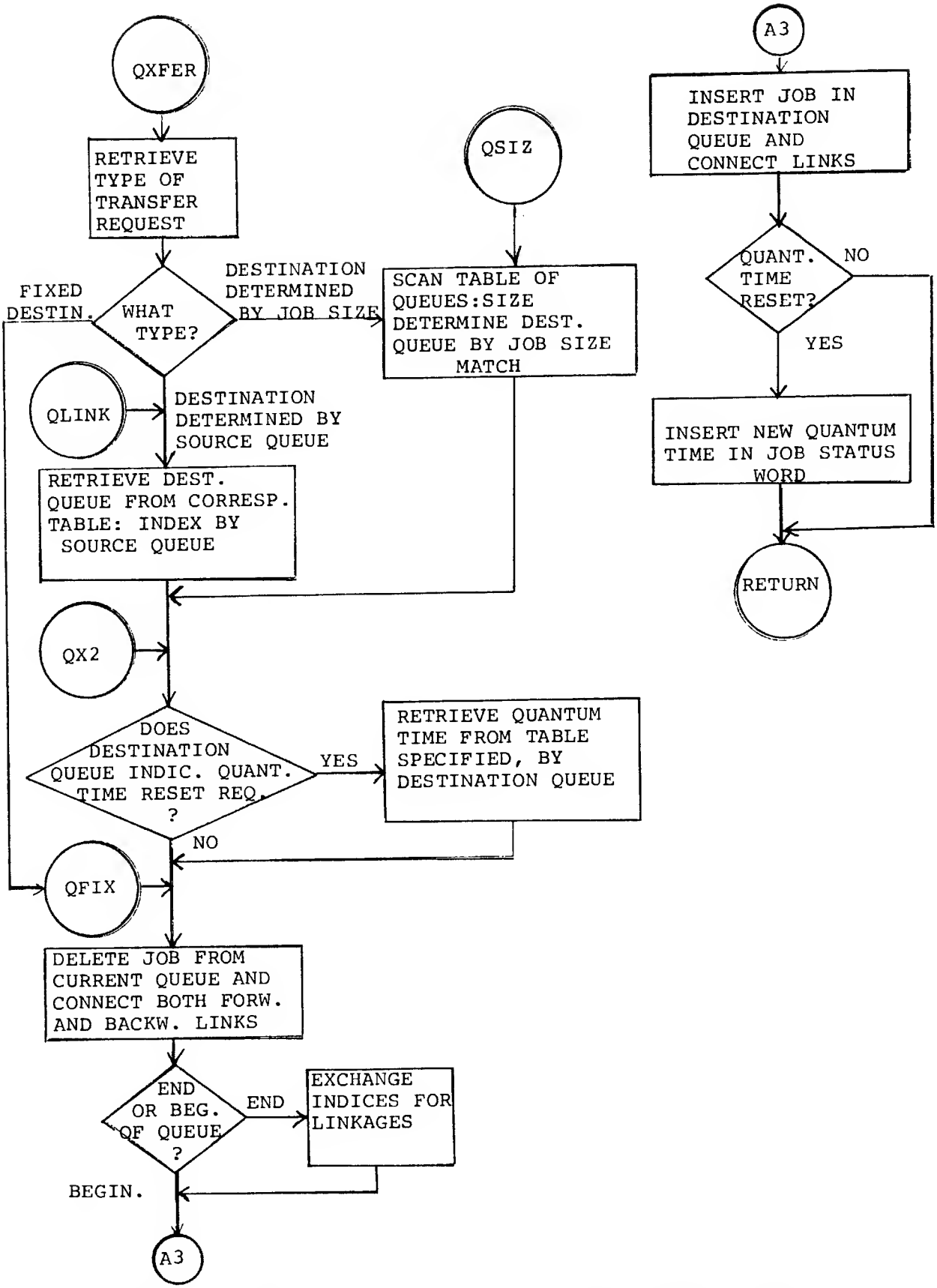



Figure 3 (Cont.) Job Scheduling, 10/50 Swapping System

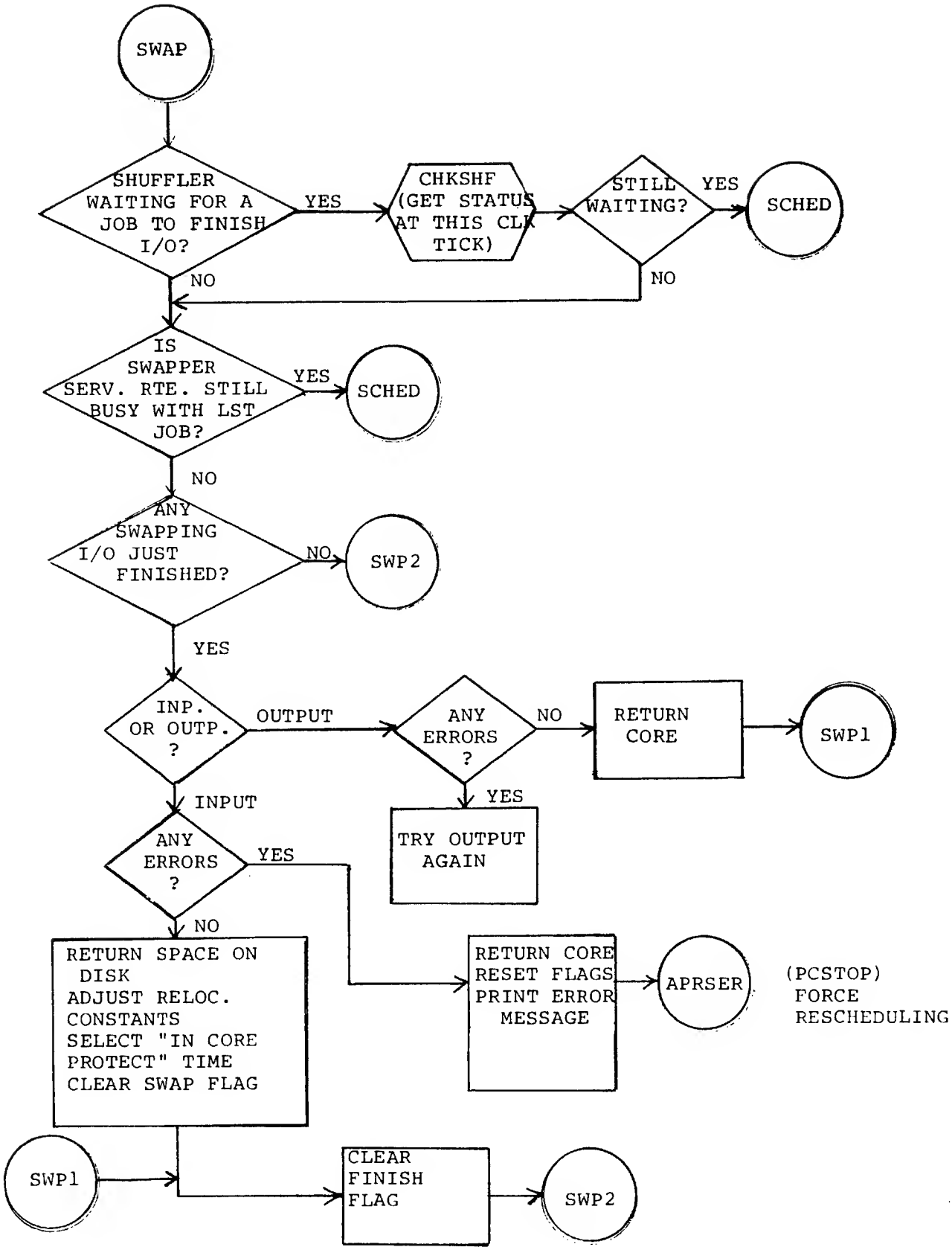



Figure 3 (Cont.) Job Scheduling, 10/50 Swapping System

Table 2

Cross Reference Listing of Job Scheduling Symbols[1]

| | |
|---|---|
| APRSER | D1, D2 |
| xxxAVAL | D4 |
| CHKSHF | F14 |
| CKJB1 | D6, F16 |
| CKJB3 | D6, F16 |
| CKJB4 | F16 |
| CKJB5 | D6, F17 |
| CKJB6 | D6, F17 |
| CKJB6A | D7, F17 |
| CKJB7 | D7, F17 |
| CKJB8 | D7, F17 |
| CKJB9 | F16 |
| CLKCSS | D1, D2 |
| CLKCSW | D5 |
| CSSDAT | D4 |
| FIT | D12 |
| FIT1 | D12, F20 |
| FORCE | D12 |
| FORCE1 | D12, F20 |
| ITEM | D3, D4, D7 |
| JOBP | D3, D4 |
| NXT0 | D2, F14 |
| NXT2 | D3, F14 |
| NXT3 | D3, F14 |
| NXT5 | D3, F14 |
| NXT7 | D3, F14 |
| NXT8 | D3, F14 |
| NXTJOB | D1, D2, D5, F14, F15 |
| NXTJB1 | F15 |
| QFIX | F18 |
| QLINK | F18 |
| QSCAN | F17, |
| QSIZ | F18, F20 |
| QX2 | F18 |
| QXFER | D2, D7, F15, F16, F17, F18 |
| RNAVAL | D3, D4, F14 |
| SCHED | D2, D7, F17 |
| SCHEDU | D2 |
| SCNOUT | F20 |
| SWAP | D2, D7, D11, F17, F19 |
| SWP1 | D12, F19 |
| SWP2 | F20 |
| TSAVAL | D4 |
| WSAVAL | D4 |
| XCKCSS | D2, F14 |

[1] A D preceding a page number indicates that a description of the item is found on that page; an F indicates that the item appears in a flow chart on that page.

# 3

# COMMAND DECODER
# (COMCON)

## AN OVERVIEW

The Command Decoder is called from the clock routine (channel 7 interrupt) whenever the counter COMCNT, which is set by the Teletype service routine (SCNSER) and indicates the number of commands waiting to be serviced, is greater than Ø. Since the Command Decoder (COMCON) is called from the clock routine, it is imperative that a command function must run to completion quickly. If the function cannot do this, the Command Decoder must set the job to a runnable status and either return immediately or delay the command for later execution.

Basically, the Decoder is divided into three parts.

1. Initial Setup and Verification

2. Command Routine

3. Cleanup and Return

The initial setup and verification portion accesses the command string typed in, verifies its legality, and performs certain procedural checks (if login is required by the command, has login been executed, or if a job number is required, is there one, etc.). It also establishes the proper linkages for the job data area, error routines, and command delay, etc.

The command routine performs the function associated with the command. In the following discussion of the Decoder, the START, R, and RUN commands are elaborated upon.

The cleanup and return portion is entered from the command routine. It restores certain information, sets the Teletype to either User or Monitor mode, and sets it to run. For new jobs, it types out the job number line. Also, it starts the typing of any messages to be output.

## ADDING A COMMAND

In addition to supplying the command routine itself, the user may only have to complete the NAMES table to add a command of his own to the Command Decoder.

The NAMES table defines the name of the command, the name of the command routine, and bit settings corresponding to those to be checked either before or after the command routine is executed.[1] It is defined by a Macro "C" setup in the following order.

| C | name of command | , | name of command routine | , | bit setting additions per Table 1 |
|---|---|---|---|---|---|

[1] See Table 1.

Examples

1. C START, START, NOPER+TTYRNU+INCORE+NOACT+NORUN

2. C RUN, RUNCOM, NOCORE+NOPER+TTYRNQ+NOCRLF+INCORE+NOACT+NORUN

One decision the user must make is whether or not to include the addition under a conditional assembly.

Table 1

COMTAB BITS

a. Bits checked before dispatching to command routine

```
NOCORE      ;NO CORE NEEDED FOR COMMAND

NOJOBN      ;NO JOB NUMBER NEEDED FOR COMMAND

NOLOGIN     ;JOB DOES NOT NEED TO HAVE BEEN LOGGED IN

NOACT       ;COMMAND MUST BE DELAYED IF JOB HAS ACTIVE DEVICES

INCORE      ;COMMAND MUST BE DELAYED IF JOB HAS CORE ASSIGNED BUT
            ;JOB IS NOT IN CORE

NORUN       ;AN IMPLIED ^C MUST BE EXECUTED BEFORE COMMAND IS
            ;EXECUTED IF JOB IS RUNNING
```

b. Bits checked after command routine has been executed

```
CMWRQ       ;REQUEUE JOB AFTER COMMAND WAIT

NODATE      ;DON'T PRINT DATE DURING JOB INTERROGATION

NOINCK      ;NO CHECK FOR JOB INITIALIZATION

NOCRLF      ;NO OUTPUT OF CARRIAGE RETURN, LINE FEED

NOPER       ;NO PRINTING OF PERIOD

TTYRNU      ;SET TTY TO USER MODE AND START JOB AFTER COMMAND
            ;RESPONSE FINISHES TYPING

TTYRNC      ;KEEP TTY IN MONITOR MODE AND START JOB AFTER
            ;COMMAND RESPONSE FINISHED TYPING

NOMESS      ;DO NOT START TTY OUTPUT
```

The following is a functional description of the Command Decoder (COMCON).

## INITIAL SETUP AND VERIFICATION

COMMAND — This routine performs for the initial setup and verification of the command string that was typed. A PUSHJ TTYCOM command sets up the job number, the byte pointer to the command string, the byte pointer to the last output character, and the address of the service data block typing command. The TTY device data block address is put on the pushdown list. A PUSHJ CTEXT scans and returns the command name. The number of characters (whether all or just part of the command) typed in are then compared to the COMTAB table of legitimate commands. If one, and only one, command matches, its index is obtained and control proceeds to COMFND. If more than one command matches, the index remains at the first reference and control proceeds to COMFND.

COMFND — This routine sets up the entry in the dispatch table DISP on the pushdown list. This table is organized and accessed in the same manner as COMTAB and contains bit settings to be checked either before command execution (e.g., NOJOBN, INCORE) or after return from command execution (e.g., CMWRQ, NOINCK). If a login procedure is required by the command, the job status word is checked to see if the job is logged in; if the required login has not been performed, control is transferred to COMER to type "LOGIN PLEASE." If a job number has been assigned, control proceeds to CHKRUN. If the command does not require a job number, control goes to COMGO. Otherwise, the job status table JBTSTS is searched for the first free job number; this number is obtained and control goes to NEWJOB. If the maximum number of jobs as set at Build time has been exceeded, a transfer to COMER prints the message "JOB CAPACITY EXCEEDED."

NEWJOB — This routine sets PROG and JDAT[1] to Ø since a new job has no core. A PUSHJ JOBINI initializes the "assign by console for TTY" bit, sets the logical name to Ø (DEVLOG in DEVDAT), and, if time accounting is part of the system, clears the incremental and total job running times and returns to CHKRUN when PROG is Ø. If the job has core, processing continues at JOBINI.

CHKRUN — This routine transfers to CHKACT if the job is runnable (if bit 1 of the job status word for this job is set). If an implied ↑C must be performed before this command can be executed, then a transfer to COMER to print "PLEASE TYPE ↑C FIRST" takes place.

---

[1] PROG and JDAT are address indicators that are used throughout the Monitor PROG is the protection-relocation register. JDAT is the address of the job data area.

CHKACT If this is not a new job, the address of the job data area is moved to JDAT. Protection and relocation information is then moved to PROG. If this is a swapping system:

1. If the job is not on disk or on its way out to disk, control is transferred to CHKCO2.

2. If the job is on disk or on its way out to disk and the job must be in core for this command, the command execution is delayed (by setting DLYCM as the command routine); control then goes to COMDIS.

If this is a nonswapping system, control goes to CHKCO2.

CHKCO2 If this command cannot be performed with active I/O devices, this routine does a PUSHJ to RUNCHK. RUNCHK delays the command and returns only if the job has stopped and there are no active devices. If it can be performed with active

CHKCO1 devices, control goes to CHKCO1. At CHKCO1, if this command does not need core, the command can be dispatched and control is transferred to COMGO; if the command does need core and core is not assigned, control is transferred to COMER to type "NO CORE ASSIGNED" (if core has been assigned, control goes to COMGO).

COMGO If the job was in command wait (CMWB = 1; this means that a command requiring core has been typed for this job, which is currently on disk - This bit is cleared when the job is brought into core), the command wait bit in the job status word is cleared. If the job was not in command wait, the "requeue job after command wait" (CMWRQ) bit in the dispatch flag entry on the pushdown list is cleared, and the CMWB bit in the job status word is also cleared. Control proceeds to COMDIS.

COMDIS This routine dispatches to the command setup routine. First, accumulator IOS is cleared for setting dispatch addresses. Then, a PUSHJ is executed to dispatch to the selected command setup routine. In case of an error, control is transferred to CERR. If there is no error, control proceeds to the command execution routine as determined by the address part of the dispatch table. In any case, return is always made to COMRET.

## COMMAND ROUTINES

To illustrate command decoding, the handling of the START, R, and RUN commands is discussed below.

START and CSTART Commands

The START command begins execution of a user's program.

START adr

The optional argument, adr, is an octal value. If it is specified, execution is begun at this address. If it is not specified, the program is begun at the starting address found in location JOBSA (right half) of the job data area.

The command routine can be called by

START

START adr

CSTART

CSTART adr

START and START adr leave the user console in user mode. CSTART and CSTART adr leave the console in Monitor mode. The setting of the mode is performed in COMRET.

OCTIN — A transfer is made to this routine to convert any optional octal address argument. OCTIN performs a PUSHJ SKPS1 to skip leading spaces, tabs, and nulls. It does not return if the previous character or the next nonspacing character is a carriage return. If there was no starting address specified, the starting address is obtained from JOBSA of the job data area and then a JRST to USTART is performed. If there was an error, a transfer to COMERA with no return is performed. The normal return is to USTART.

USTART — This routine is part of the RUNCSS section of APRSER in the Monitor. It sets the job state to be scheduled to run with the specified starting address, including the PC flags. If the old program counter (PC) indicates that it was not in user mode, the user's accumulators are moved to the dump accumulators, the PC and APR flags are preserved in JOBOPC in the job data area. If it was in user mode, the new PC is set to user mode. The new PC is stored in JOBPC of the job data area.[1] The error and wait status bits are set and a transfer (JRST) to TTYSET occurs.

This routine is called only when the job is in core and after the job has been safely stopped in one of three states:

1. PC is in user mode;
2. Job is in a wait for a sharable device or I/O wait;
3. Job is just about to return to the user mode from a UUO call.

[1]The old PC and APR flags are preserved in JOBOPC of the job data area.

TTYSET TTYSET, which is part of the SCNSER section of the Monitor, sets the user's Teletype to initial conditions by clearing

1. DDTM (DDT mode bit)
2. NIO (stop all I/O)
3. IOSUPR (suppress all I/O until next input or initialization)
4. USRB (set TTY to user mode when I/O finishes)
5. TTYIOW (TTY input wait bit)
6. IOW (input wait bit)

This routine also sets the data mode to ASCII line mode in case the user types a line before the program does an initialization and transfer to MIS1; at that point, the DEVIOS entry is modified and a POPJ is performed, then control is transferred to COMRET.

R Command

The R command is in the form

```
R cuspname core
```

It performs a

```
RUN SYS: cuspname
```

by setting the device name to SYS and transferring to the second instruction of the RUNCOM routine (the command following the PUSHJ CTEXT1 as described under "RUN Command" below).

RUN Command

The RUN command is in the form

```
RUN log-dev filename.ext  proj,prog  core
```

where

log-dev is the logical or physical name of the device
filename is the name of the program
.ext is the extension name (optional)
proj, prog is the project-programmer number if the file is located in other than the user's area of disk (optional)
core is the amount of core to be assigned (optional)

The RUN command loads a core image with the name filename(.ext) from a retrievable storage device (log-dev) and starts it at a location specified within the file. The minimum amount of core required to load the file is allocated. After the file is loaded, core is reallocated if the optional argument, core, is specified or if the file was saved with a core argument. If both were specified, the core argument of the RUN command takes precedence.

RUNCOM The first function performed by this routine is to PUSHJ CTEXT1 to obtain the device name from the command string. The address RUNJOB is saved in accumulator IOS for a return and is also put on the pushdown list. If the job does not have core, the input byte pointer to the command string is saved on the pushdown list, the device name is saved on the pushdown list, and the TTY "assigned by program" bit (ASSPRG) in DEVMOD of the device data block is cleared; just enough core for the job data area is requested and a PUSHJ CORE$\emptyset$ is performed. If the job does have core, control is transferred to RUNCO1.

CORE$\emptyset$ CORE$\emptyset$, in the CORE1 section of APRSER in the Monitor is called by the CORE command, the core shuffler, and the RUN command. The job is moved, if necessary to satisfy the request. The accumulators PROG and JDAT are set to the new core assignment on either an error ($\emptyset$ is assigned when either the job has active I/O or there is not enough core) or an OK return. The error return, in this case, is a NOP (an JFCL with no flags, which is the fastest NOP). The device name, input byte pointer, and the address of the Monitor job are restored.

RUNCO1 If this is not a swapping system, a test is made to see if PROG had been set. If PROG was $\emptyset$, the message "CORE IS FULL" is typed. If PROG was set, a transfer is made to SGSET.

If this is a swapping system and core was not assigned, a JRST to DLYCM delays the command; if core was assigned, a transfer is made to SGSET.

SGSET SGSET is in the SAVGET section of APRSER in the Monitor. It scans the command string arguments of SAVE, GET, RUN, and R commands, and stores them in the job data area, which must be in core. It also stores any extension names (such as .SAV and .DMP), the project-programmer number (optional), and the optional core argument. Upon return from the core conversion, if there are no errors or there was no core argument, it saves the device name, schedules the Monitor job (RUNJOB, GETJOB, or SAVJOB), and does a JRST to MSTART to start the job with the PC in Monitor mode; if there was an error, a transfer is made to COMERA.

MSTART MSTART is the last portion of the USTART routine. It stores the new PC, clears the error and wait status bits, and performs a JRST to TTYSET (discussed under the START command).

DLYCM DLYCM delays the execution of a command until the job is in core memory. There are three entry points to this routine: DLYCM, DLYCM1, and DLYCM2. DLYCM is a PUSHJ to DLYCOM, following which it proceeds to DLYCM1. DLYCM2 is assembled

if this is a swapping system and provides a POP of the pushdown list and a jump to DLYCM1. DLYCM1 proceeds to the remaining portion of this routine.

DLYCOM — This routine is in the RUNCSS portion of APRSER. It sets the command wait bit in the job status word. It requeues a job which has had a command typed which needs core and which is either on the disk or is in core and has active devices; it does this with a PUSHJ to REQUE.(REQUE is the last portion of the routine SETRUN, in RUNCSS, and is conditionally assembled for a swapping system. It sets the requeueing flag and places it in the job number bit assignment of register QJOB to be used by the scheduler and then returns to DLYCOM, which in turn goes to DLYCM1).

DLYCM1 — DLYCM1 adjusts the pushdown pointer by POPing three times, putting the address of a POPJ pdp instruction on the pushdown list, putting the line number into accumulator TAC1, and performing a JRST to TTYCM.

TTYCM — This is the second entry point to TTYCOM, in the SCNSER section of the Monitor. It sets DEVDAT to the address of the Teletype on which the command was typed and returns to COMRET.

SETRUN — This routine, in the RUNCSS section of APRSER, sets the job status run bit and is called by the Scanner S rvice routine when the TTY Monitor command response finishes. This action is enabled by calling TTYUSR or TTYURC in SCNSER. If this is a swapping system, processing continues with the REQUE portion, discussed under DLYCOM. If this is not a swapping system, control transfers to NULTST to see if the null job is running.

CLEANUP AND RETURN (COMRET)

This is the return from the command setup routine. It completes the function of the Command Decoder by adjusting certain bit indicators, setting the Teletype to either user or Monitor mode, and requeueing the job.

1. The command flags and the TTY data block address are restored from the pushdown list (TTY DDB goes to DEVDAT).

2. The sign bit of Table TTYTAB (the Teletype translator table in the SCNSER section of the Monitor) is turned off to indicate that the command has been processed.

3. A 1 is subtracted from COMCNT to reduce the number of commands waiting to be processed.

4. If there is no job number and one is required, bits NOINCK (no check for job initialization) and ERRFLG (command error) are set.

5. If the job initialization bit (JNA) of the job status word is already set, or if the suppress job initialization check bit is on (NOINCK), control goes to step 11; otherwise, the JNA bit of the job status word is set.

6. A PUSHJ TTYATI (in SCNSER section of the Monitor) is executed to attach the Teletype to the job.

7. A PUSHJ INLMES (in COMCSS section of APRSER in Monitor) prints the job number.

8. The job number is converted by the routine RADX10 (in ERRCON section of APRSER in Monitor) to print number in decimal.

9. A PUSHJ PRSPC prints four spaces.

10. The system configuration name, four spaces, and today's date are printed.

11. PCRLF: If an error message occurred, a PUSHJ PRQM appends and prints a question mark.

12. If a carriage return, line feed, are to be printed, a PUSHJ CRLF is performed.

13. If a period is to be printed, a PUSHJ PRPER is performed.

14. If there is an error or a job number has not been assigned, the next few instructions, which set the job to run in either the Monitor or user mode, are bypassed and control goes to step 17 (PCRLF1).

15. If job is to run in user mode (TTYRNO = 1), a PUSHJ to TTYUSR is performed.

16. If job is to run and remain in Monitor mode (TTYRNC = 1), a PUSHJ to TTYORC is performed.

17. PCRLF1: If there is a message to type (NOMESS = Ø), a PUSHJ to TTYSTR is performed.

18. If this is a swapping system and a job has to be requeued after being in command wait, a JRST to REQUE is performed. Otherwise, a POPJ is performed to exit.

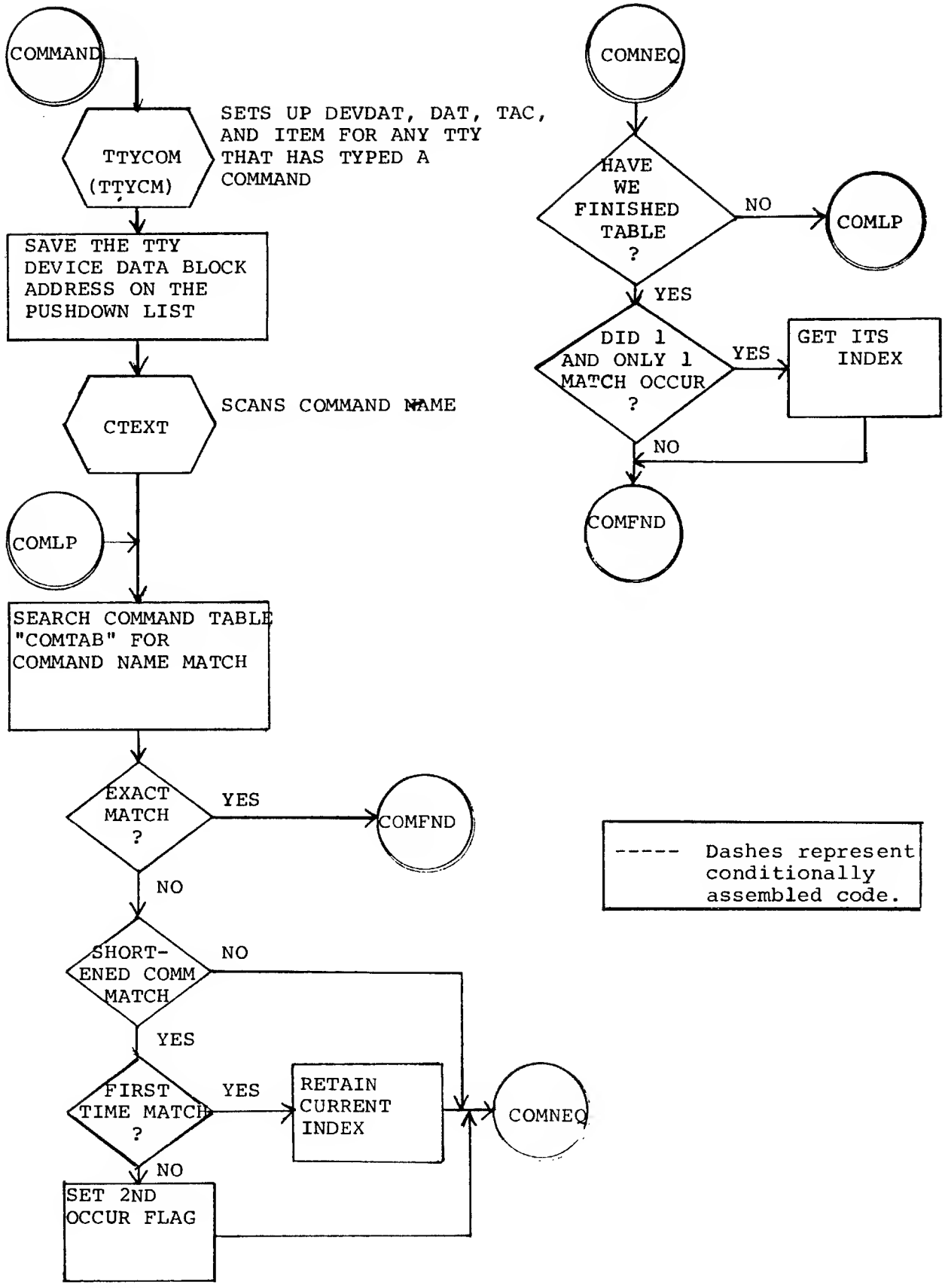



Figure 1. Command Decoder Flow Chart - Initial Setup and Verification

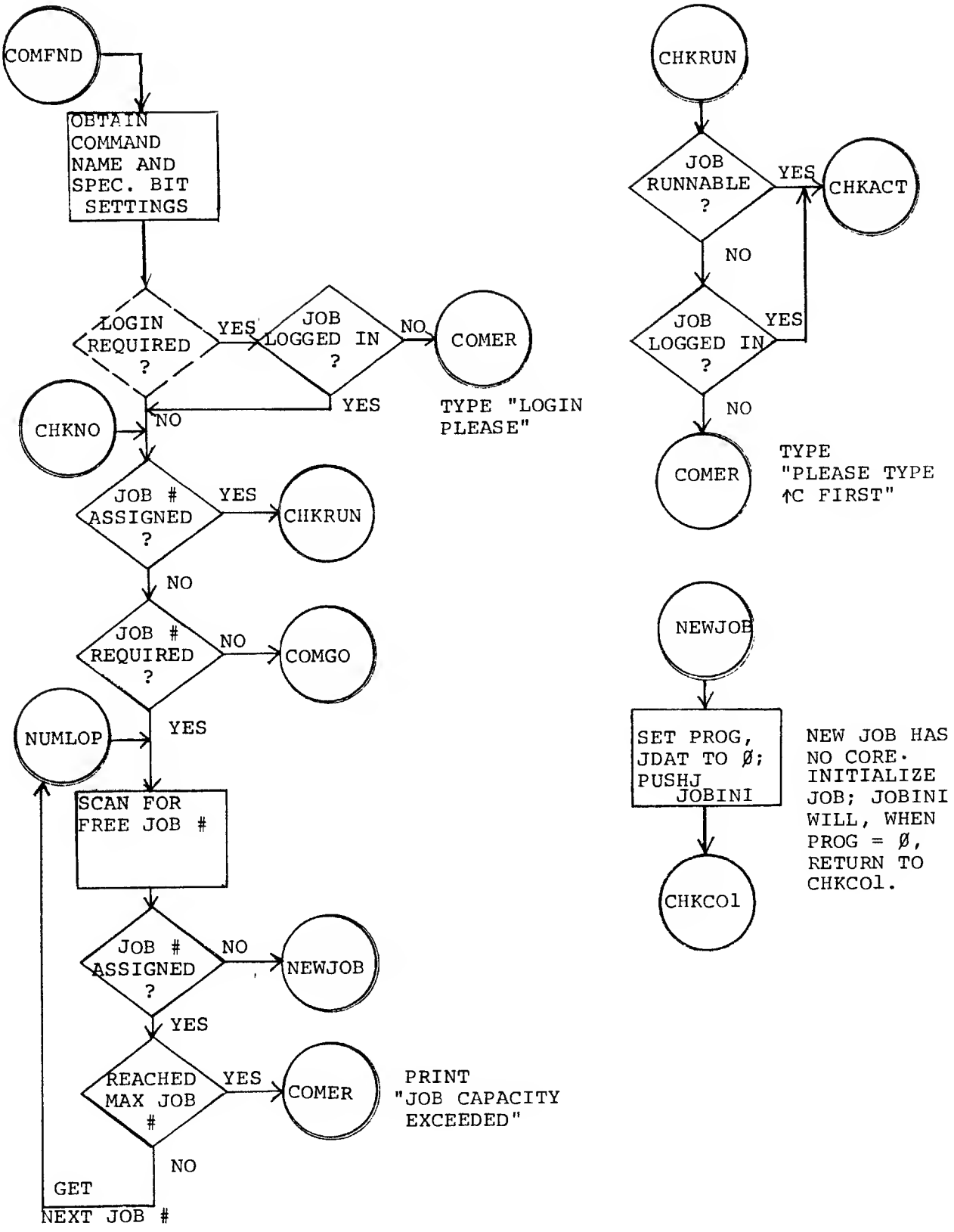



Figure 1 (Cont.) Command Decoder Flow Chart - Initial Setup and Verification

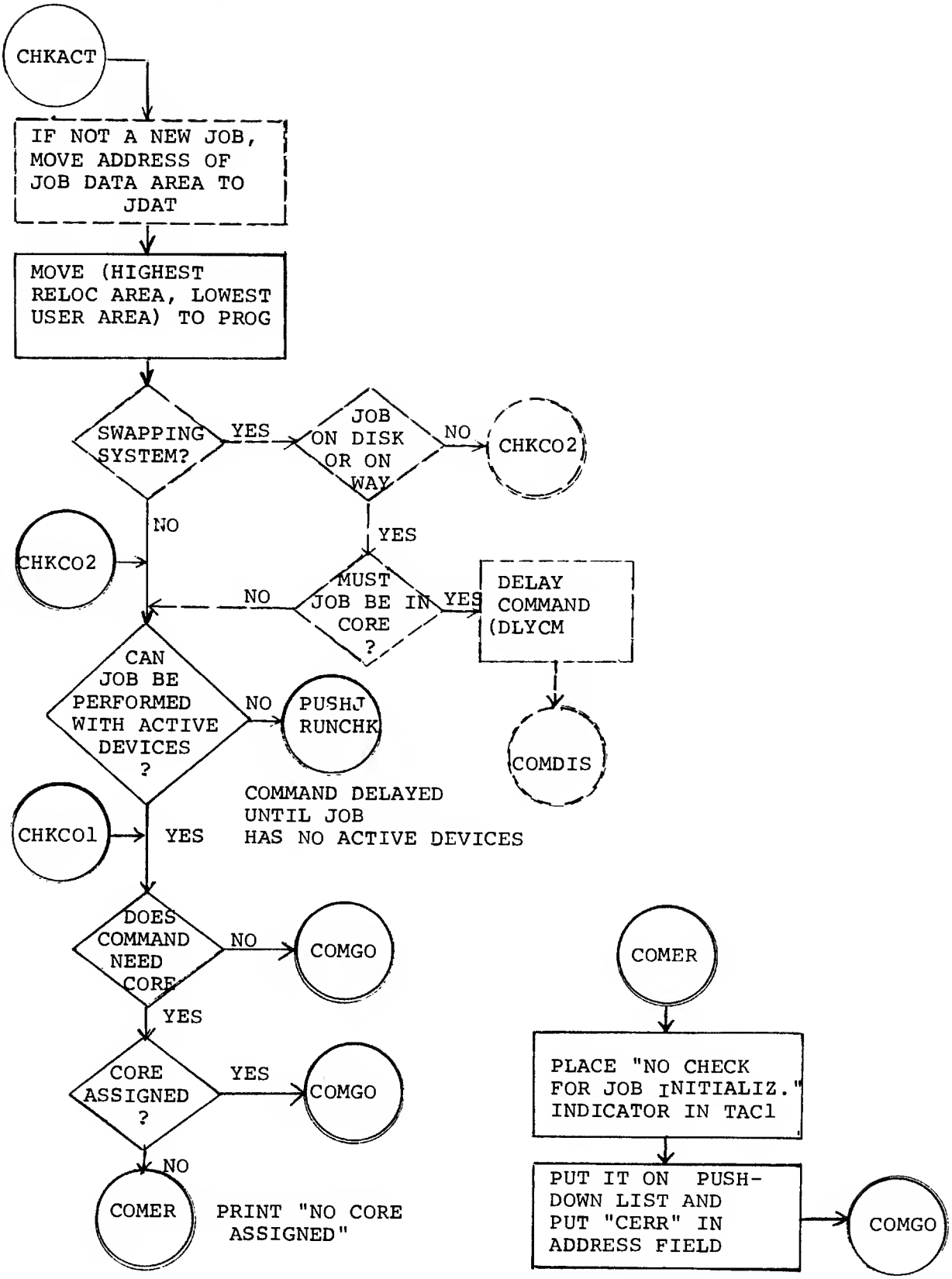



Figure 1 (Cont.) Command Decoder Flow Chart - Initial Setup and Verification

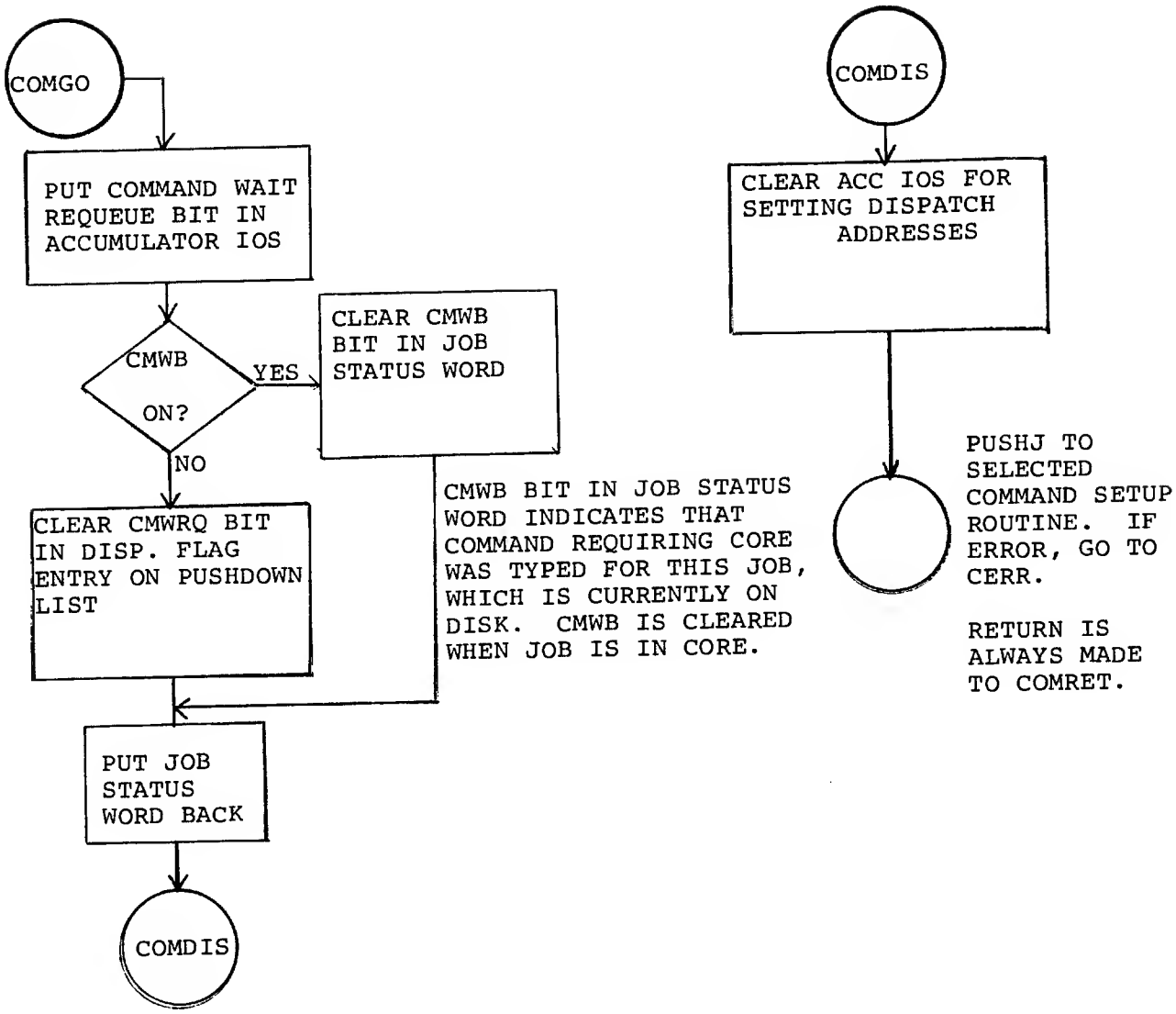



Figure 1 (Cont.) Command Decoder Flow Chart - Initial Setup and Verification

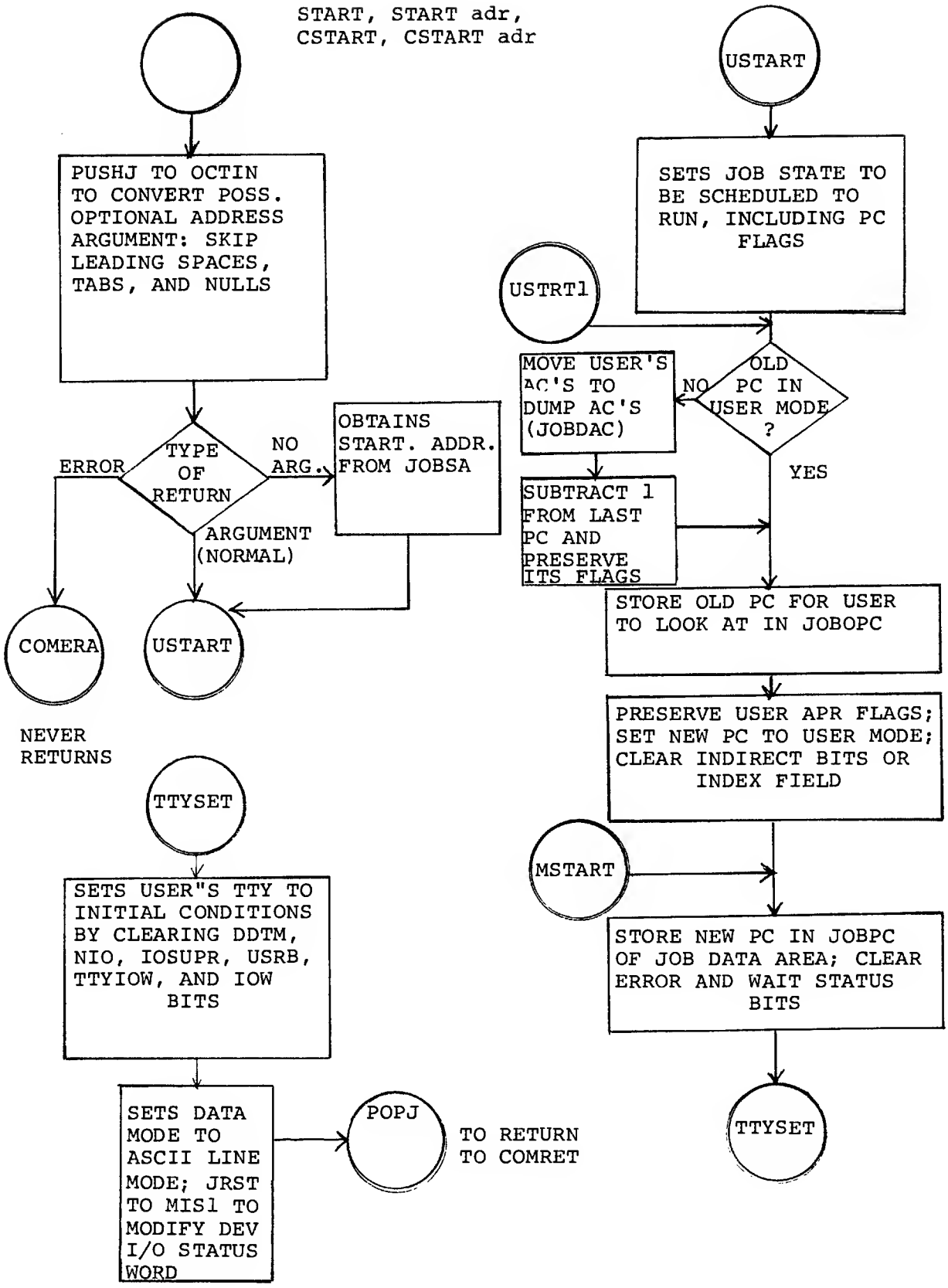



Figure 2. Command Decoder Flow Chart - Command Routines for START, CSTART Commands

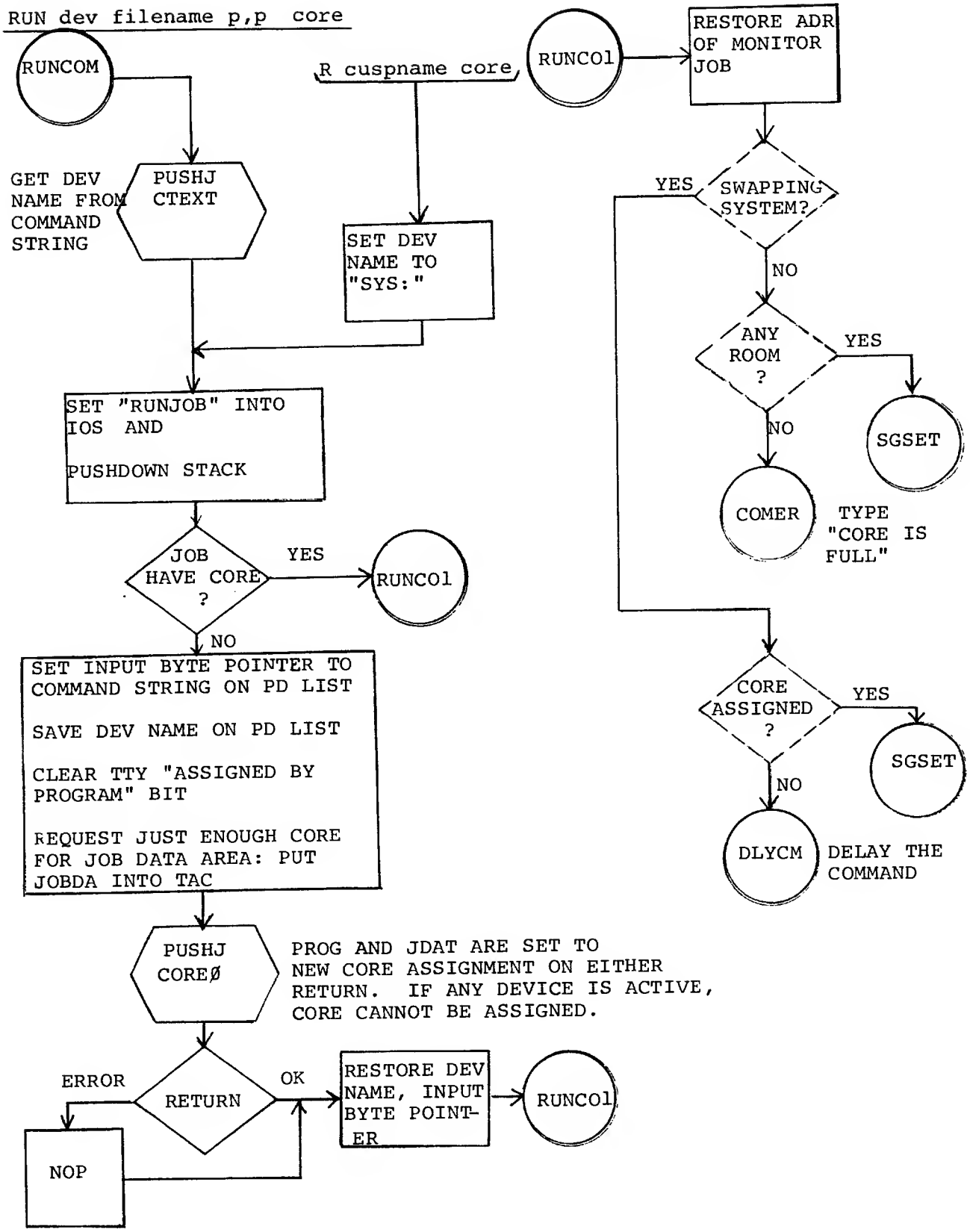
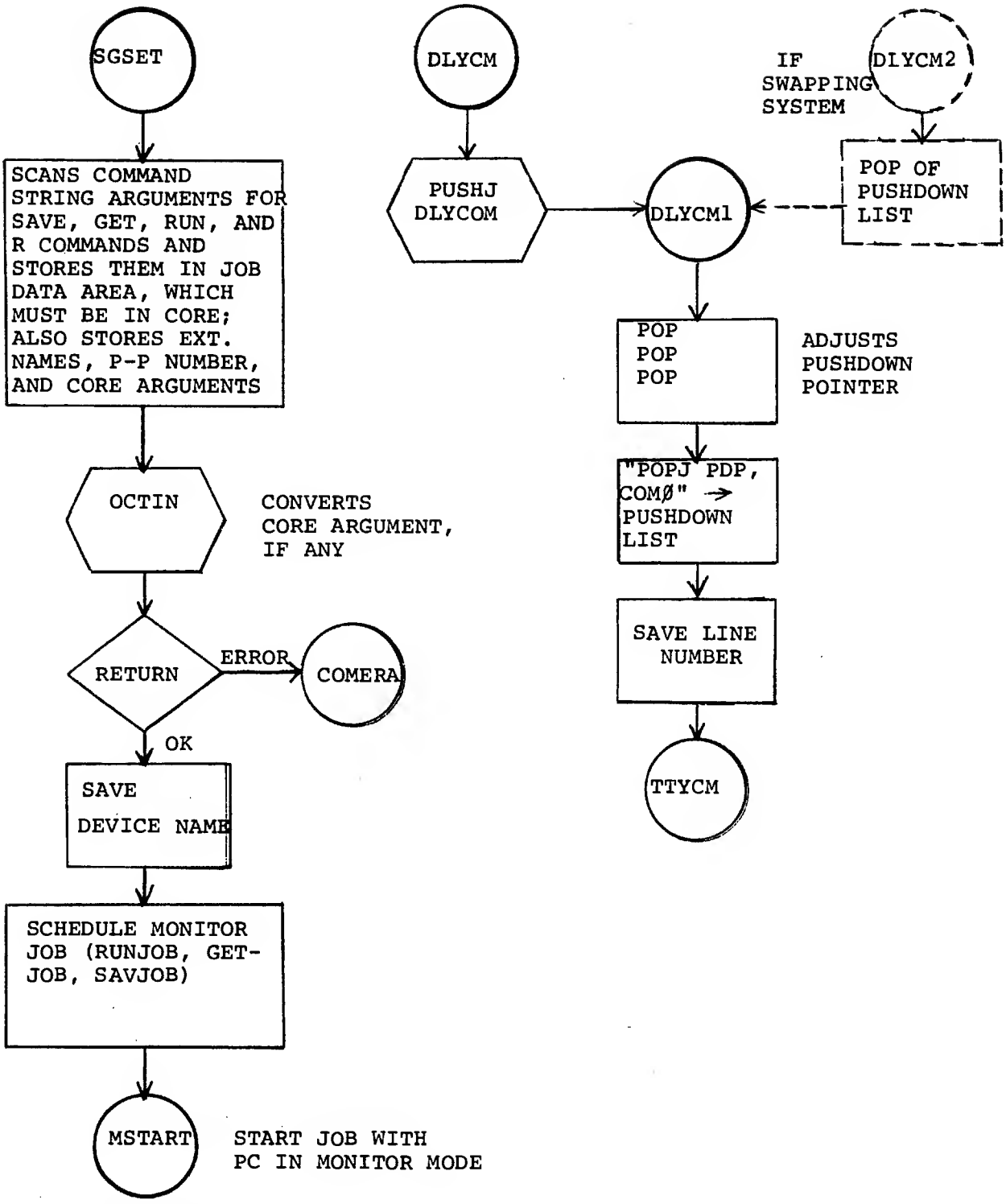



Figure 3. Command Decoder Flow Chart - Command Routines for R, RUN Commands




(SEE PAGE 14)

Figure 3 (Cont.) Command Decoder Flow Chart - Command Routines for R, RUN Commands

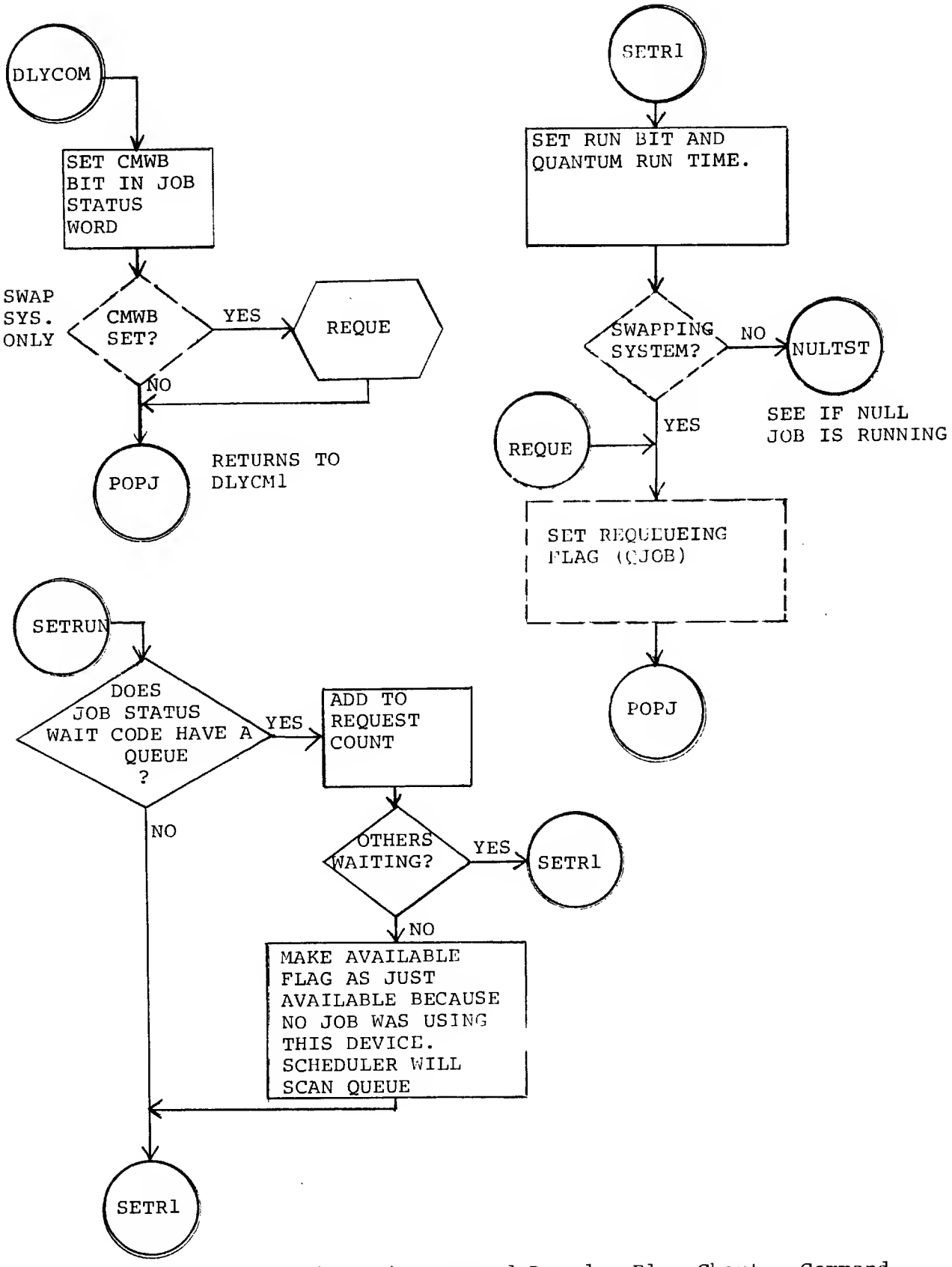



Figure 3 (Cont.) Command Decoder Flow Chart - Command Routines for R, RUN Commands

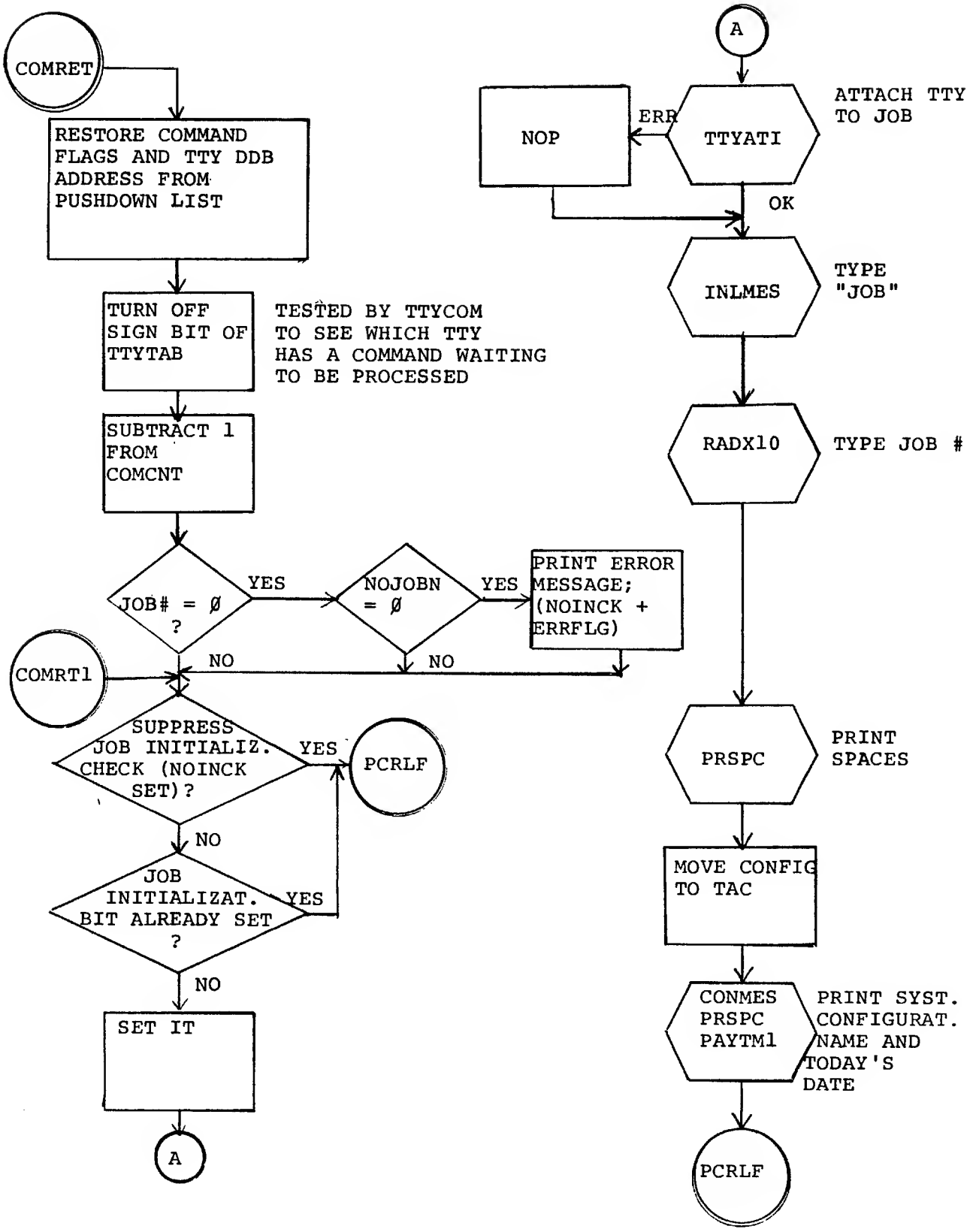



Figure 4. Command Decoder Flow Chart - Cleanup and Return

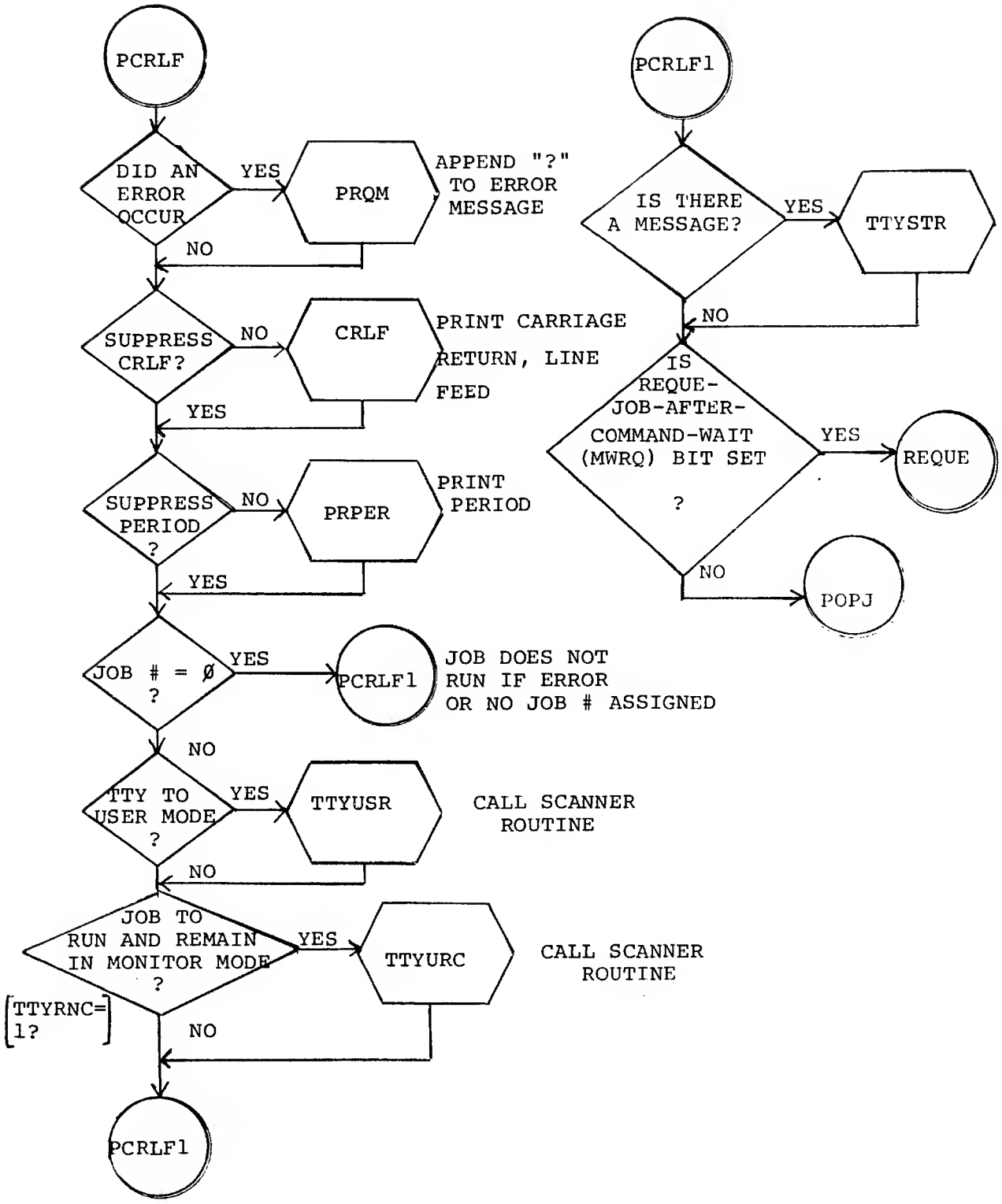



Figure 4 (Cont.) Command Decoder Flow Chart - Cleanup and Return

Table 2

CROSS REFERENCE LISTING OF SYMBOLS IN COMMAND DECODER[1]


| | |
|---|---|
| CHKACT | D4, F12 |
| CHKCO1 | D4, F12 |
| CHKCO2 | D4, F12 |
| CHKNO | F11 |
| CHKRUN | D3, F11 |
| CMWRQ | D2 |
| COMFND | D3, F11 |
| COMDIS | D4, F13 |
| COMER | F12 |
| COMGO | D4, F13 |
| COMLP | F10 |
| COMMAND | D3, F10 |
| COMNEO | F10 |
| COMRET | D9, F18 |
| COMRT1 | F18 |
| CONMES | F18 |
| CORE0 | D7, F15 |
| CRLF | D9 |
| CTEXT | F10, F15 |
| DLYCM | D7, F16 |
| DLYCM1 | D8, F16 |
| DLYCM2 | F16 |
| DLYCOM | D8, F17 |
| INCORE | D2 |
| INLMES | D9, F18 |
| MSTART | D7, F14 |
| NEWJOB | D3, F11 |
| NOACT | D2 |
| NOCORE | D2 |
| NOCRLF | D2 |
| NODATE | D2 |
| NOINCK | D2 |
| NOJOB | D2 |
| NOLOGIN | D2 |
| NOMESS | D2 |
| NOPER | D2 |
| NORUN | D2 |
| NUMLP | F11 |
| OCTIN | D5, F16 |
| PAYTM1 | F18 |
| PCRLF | D9, F19 |
| PCRLF1 | D9a, F19 |
| PRPER | D9 |
| PRQM | D9 |
| PRSPC | D9, F18 |
| RADX10 | D9, F18 |
| REQUE | D9a, F17 |
| RUNCOM | D7, F15 |
| RUNCO1 | D7, F15 |
| SETRUN | D8, F17 |
| SETR1 | F17 |
| SGSET | D7, F16 |
| TTYATI | D9, F18 |
| TTYCM | D8, F10 |
| TTYCOM | D8, F10 |
| TTYORC | D9 |
| TTYRNC | D2 |
| TTYRNU | D2 |
| TTYSET | D6, F14 |
| TTYSTR | D9a, F19 |
| TTYUSR | D9 |
| USTART | D5, F14 |
| USTRT1 | F14 |


[1] A D preceding a page number indicates that a description of the item is found on that page; an F indicates that the item appears in the flow chart on that page.

# 4

## APR AND CLOCK INTERRUPT ROUTINES

The Arithmetic Processor (APR) is assigned to the highest priority channel not in use; a BLKI or BLKO is placed in the channel's interrupt location (40 + 2j). Occasionally, one or two high-speed devices, such as a card reader, are also assigned to this same channel; however, it is required that the number of such devices be kept to a minimum. The reader may wonder why the APR is assigned to such a high priority if scheduling (or, more precisely, changing users) must occur when no interrupts are in progress to ensure that the hardware ACs are the user's rather than the Monitor's. The reason for this high priority assignment is that, in addition to the 60-cycle clock, the APR also traps on error conditions which can occur in the Monitor interrupt routines as well as at Monitor UUO level and user level. Thus, error interrupts must be handled at the highest possible priority level lest the user be blamed for an error in the Monitor. Also, the APR interrupt routine requests a secondary interrupt on the lowest priority channel (called the CLK channel) to perform job scheduling and job switching on a periodic basis. The device CLK does not actually exist as hardware, but is a creation of the software. Figure 1 illustrates the relationship between the APR channel and the CLK channel.

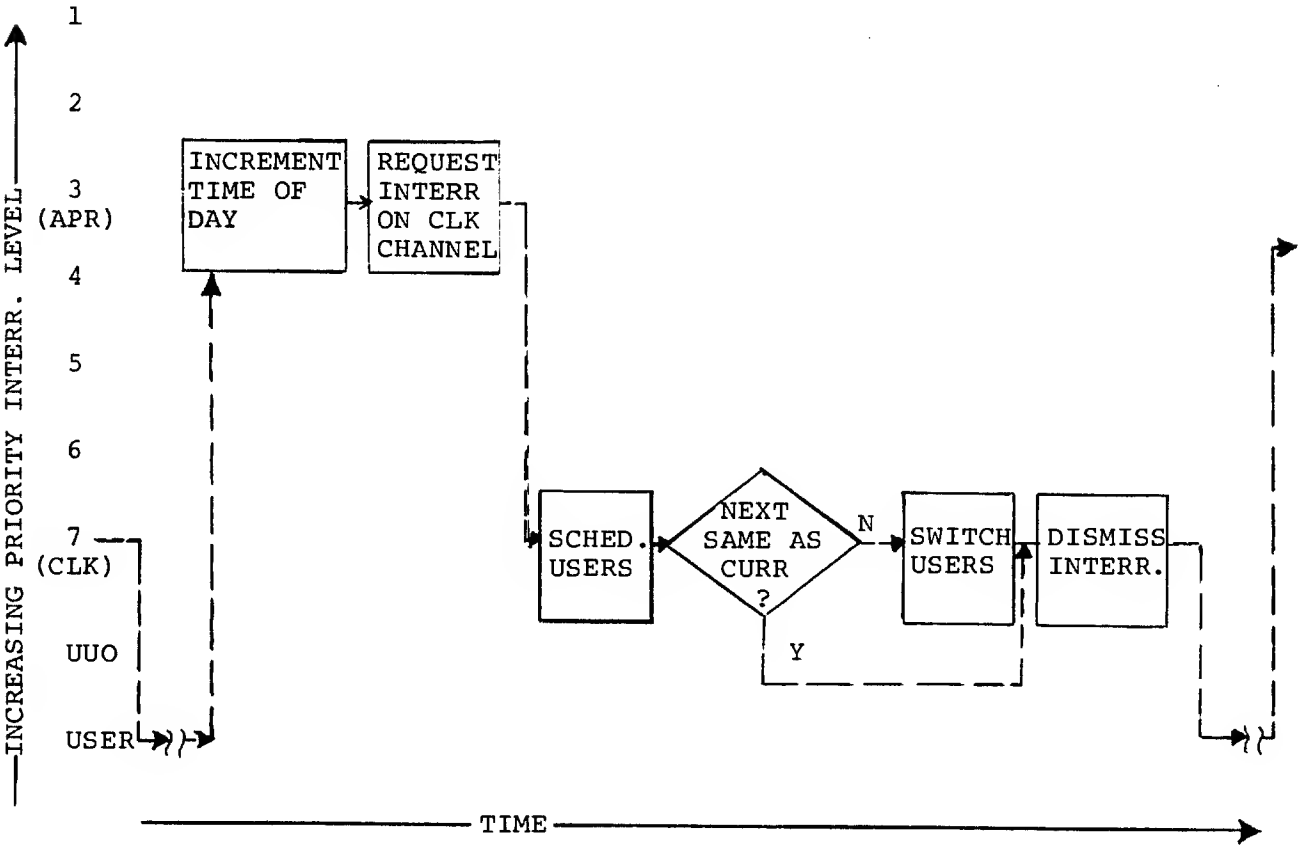



Figure 1. Relationship Between APR and CLK Channels

## APR INTERRUPT ROUTINE

APRINT

APRINT is an interrupt level routine which is entered from an interrupt on channel 3 (standard assignment). Location 46 (40 + 2j) contains a JSR CH3.

```
CH3:  Ø    ;APRCHL=CH3
      JRST APRINT
```

The interrupts on this channel are caused by any APR allowable interrupts (which are, at the least, the clock, illegal memory - ILM, nonexistent memory - NXM, and pushdown list overflow - PDOVF. If other devices have been assigned to channel 3, System Builder provides a transfer to test for them.

If this is a clock time interrupt, the time of day and uptime counters are incremented, and the following flags are set.

1. The APR clock tick flag (TIMEF).
2. The clock-forced interrupt flag (CLKFLG).
3. Request an interrupt on CLK channel (REQCLK).

If the user is enabled for any flags (including clock), a transfer is made to APRER; otherwise, the clock flag is cleared and the interrupt is dismissed indirectly to APRCHL.

APRER

This routine checks for interrupts other than the clock interrupt. If the Program Counter (PC) is not in user mode, or it is in user mode but the user is not enabled for this interrupt, a transfer is made to APRER2 to print an error message and stop the job. Otherwise, the address of the current job data area is obtained, the PC and the APR conditions are stored in the job data area, and the user's trap-answering routine address is obtained to dismiss to APRER3.

APRER3

This routine clears the PC Change and AROVF flags, clears all error flags which can cause interrupts (except for the clock flag), and dismisses indirectly to APRCHL.

APRER2

This routine transfers to APRER3 if the error condition is a PDOVF condition. Otherwise, the error PC is placed in APRPC, the error flags are stcred, and the clock interrupt flag (CLKFLG), rescheduling-needed flag (SCHEDF), and the request-an-interrupt-on-clock-channel flag (REQCLK) are set. If this is a nonexistent memory condition, a determination is made as to whether it is the PC that is at fault (this causes a dismissal to the CLKINT routine) or a data reference error (this causes a dismissal to the interrupted address); if this is not a nonexistent memory condition, the interrupt is dismissed properly by transferring to APRER3.

INTERRUPT CAUSED BY CLOCK,
ILLEGAL MEMORY REFERENCE, NONEXISTENT
MEMORY REFERENCE, PUSHDOWN LIST OVERFLOW, ETC.

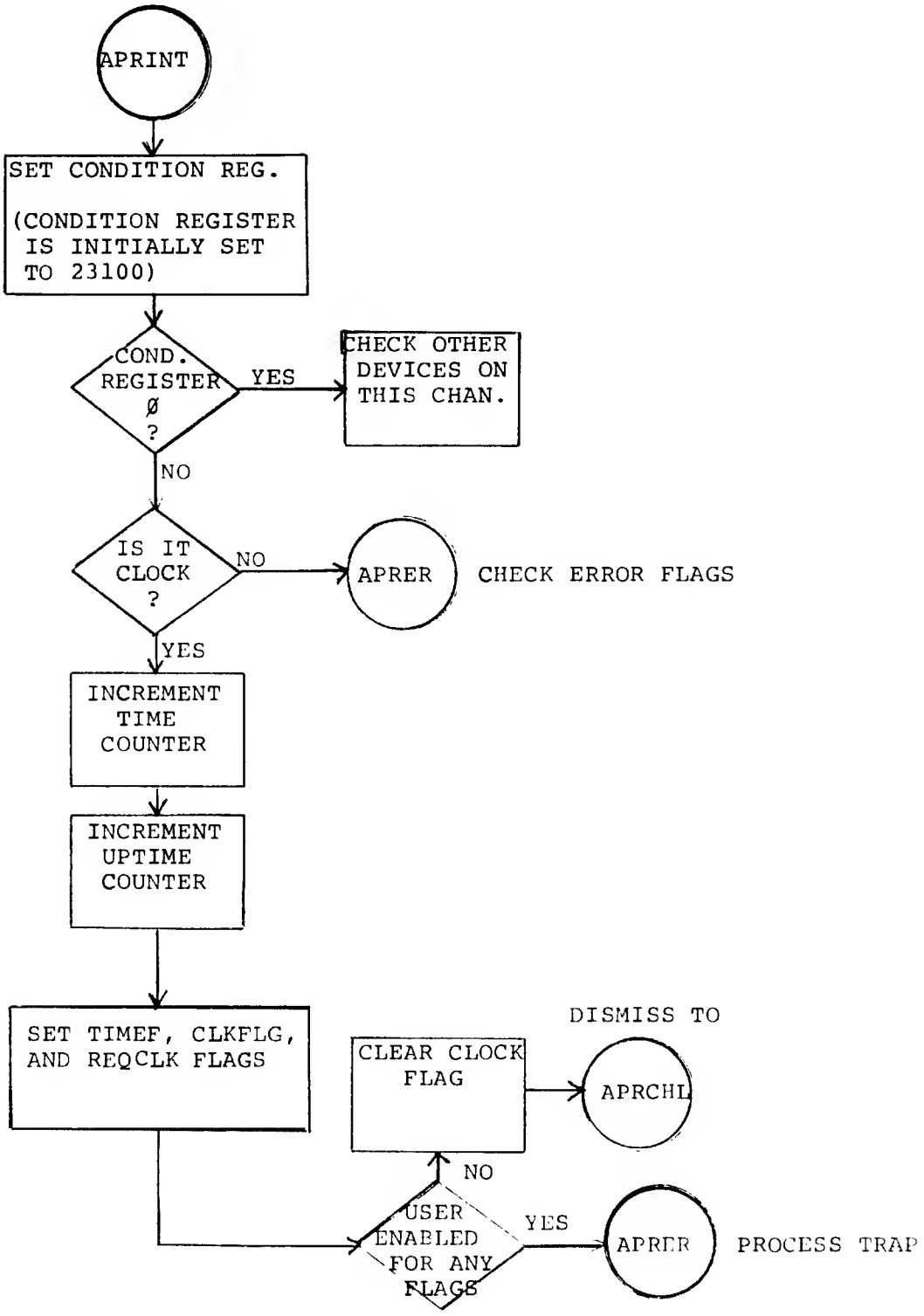



Figure 2. Flow Chart of APR Interrupt Routine

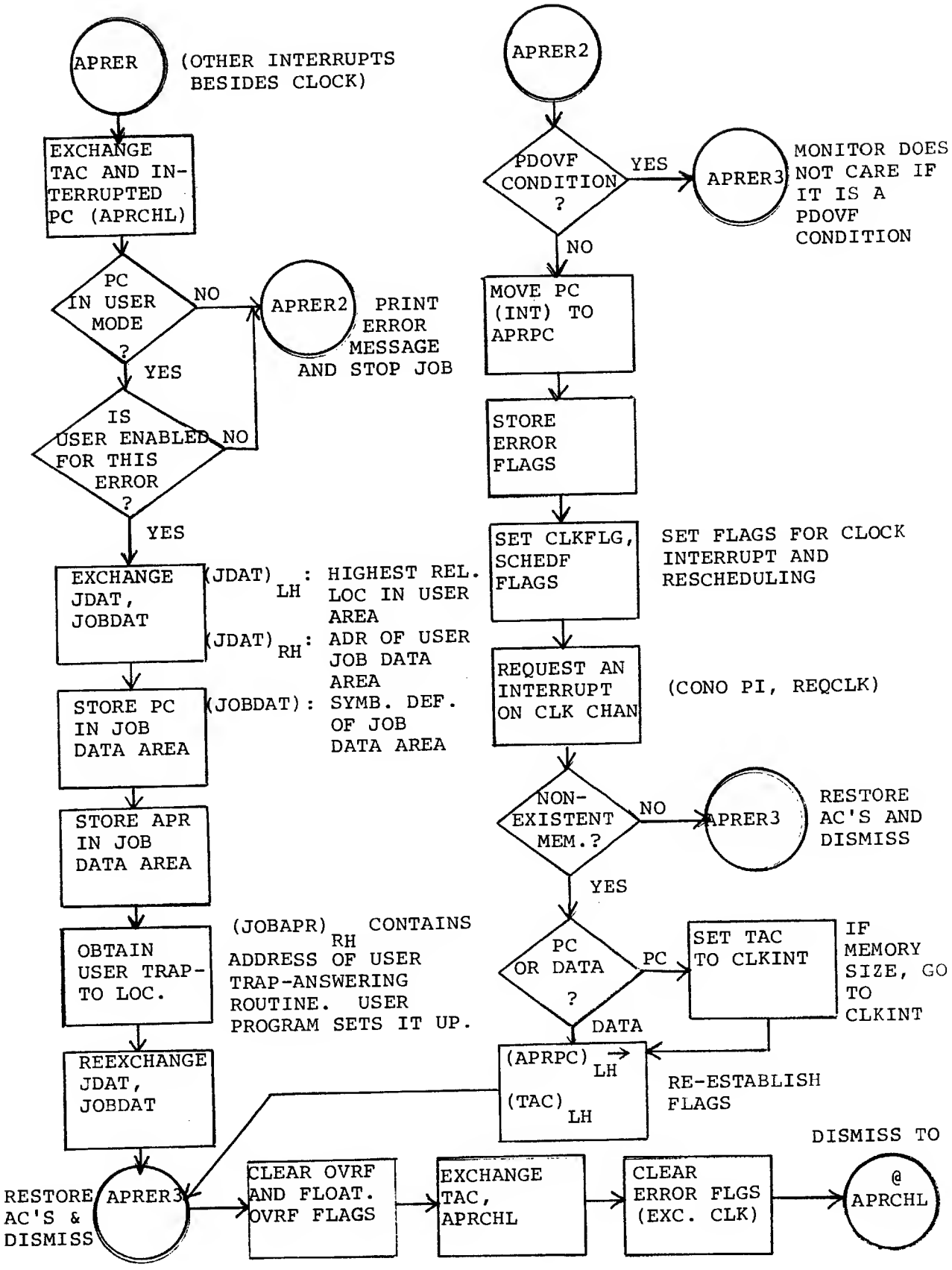



Figure 2 (Cont). Flow Chart of APR Interrupt Routine

## CLK INTERRUPT ROUTINE

The clock interrupt service routine performs the following actions on a regular basis.

1. Processes clock queue requests.
2. Calls console Monitor Command Decoder.
3. Calls the Core Shuffler.
4. Calls the Scheduler.

If the current job is in Monitor mode, the above steps are delayed until the current job enters a stoppable state such as

1. Job goes into a wait for a busy sharable device.
2. Job goes into a wait for I/O to complete.
3. Control is about to return to user mode.

The clock service routine is entered at the UUO level.

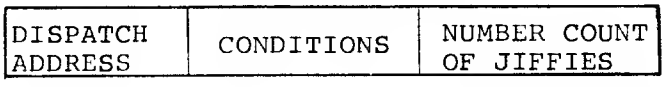
A functional description of the clock interrupt service routine follows.

CLKINT — This is the interrupt level clock channel response. If the CLKFLG has not been set, other devices on this channel are checked. If CLKFLG has been set and the current job is in user mode or this is a rescheduling situation (SCHEDF), a transfer is made to SAVPC; if the current job is not in user mode and this is not a rescheduling situation, the interrupt is dismissed by transferring indirectly to CLKCHL.

SAVPC — This routine saves the PC in USRPC (in the protected part of system data storage for current job) and saves the 17 accumulators in the dump accumulator area. It also sets up the pushdown list (JOBPDL or JOBPD1, which is the location before the pushdown list) which is used by the Monitor for UUO's and which is found within the job data area. If this is an error interrupt, control proceeds to APRILM (via a PUSHJ) - this indicates an illegal memory reference; otherwise, control goes to RSCHED.

RSCHED — This routine determines if a clock tick (TIMEF) has occurred since the last call. If not, control proceeds to CIP6 to just reschedule. If a clock tick has occurred, control proceeds to a conditionally assembled piece of code (assembled if time accounting is part of the system) which increments the total run time. Control next goes to a midnight check. Midnight check is that part of time accounting which resets or updates the time, day, or month, as appropriate. Control then proceeds to CIP2.

CIP2 — This routine is responsible for processing timing requests, such as those for sleep, rewind mag tape, etc. If the list of such requests is completed, control goes to CIP5; otherwise, decrement timing request. If time has not expired, continue the scan, returning to check if the list of requests has been completed; if the time has expired, the priority interrupt flag (PIF) is turned off, the last entry is moved into the expired jobs position, the list is decreased by 1, and the PIF is turned back on. CIPWTM (CIPWTM1 is actually used in the routine but the leading six characters only are used) is the table tag for this request list, which has the structure




CLOCK is a 36-bit byte pointer referring to CIPWTM1. A PUSHJ to the dispatch address is made (to a timing request routine). If there are more requests, return to

process them; otherwise, control goes to CIP5.

CIP5 This routine decrements the hung I/O device time and transfers to DEVCHK (via a PUSHJ) to check for hung I/O devices. COMCNT is a counter used to indicate the number of commands typed in but not decoded; it is set by the routine SCNSER and decremented by the Command Decoder (COMCON). COMCNT is now checked to see if there are any commands to process. If so, the Command Decoder (COMCON) is called by a PUSHJ. A check of COMCNT is performed every clock tick. Control passes to CIP6.

CIP6 The scheduler is called with a PUSHJ NXTJOB, which may result in scheduling, swapping, and queue reviewing; it returns an item number representing the new job to be run. Flags TIMEF, APRERR, CLKFLG, and SCHEDF are reset. If the next job to be run is the same as the previous job, a transfer is made to CIP8 to restore the accumulators and dismiss. If the jobs are different and the old job was the null job, a transfer is made to CIP7 so that the software state is not saved; otherwise, the protected part of the job data area is moved to the user's area and control goes to CIP7 (or NULJOB).

CIP7 This routine restores the software state of the new job and is also entered from SYSINI as NULJOB (with item number = Ø). JBADR is a table, with an entry for each job containing the relative maximum address in the left half and the absolute starting address in the right half; this table is used for relocation and protection. If the starting address of the user's job data area is equal to the starting address of the user area, JOBDAT is set to the location of the job data area of the current job. Certain conditions (per APRNUL) are turned off and the user APR interrupts are disabled. If the new job is not the null job, protection is set for it. Protection is also set in the job data area so that the user can look at it. The relocation and protection register is set. The protected part of the job data area is moved to the Monitor area. All APR bits (except PDOVF, ILM, NXM, CLOCK, PC CHNG - for PDP-6 use only -, and AROVF) are masked out and stored in the user-enabled CPU flags register APRIN1. The system is then assured that PDOVF, ILM, NXM, and CLOCK are enabled and all of the above are stored in APRCON, the system APR CONSO interrupt location. The arithmetic overflow condition is then restored. Control goes to CIP8.

CIP8 This routine restores the hardware state of the current job. If no job data area is set, a null job data area is set. The dump AC's are restored and the channel is dismissed.

NULJB — NULJB runs when no other job runs. If a Monitor checksum is a requirement of the system, it is checked. Then the PC is set to 1 and an accumulative count in kept in Ø (for display purposes). A dismiss to 1 occurs if an interrupt is in progress.

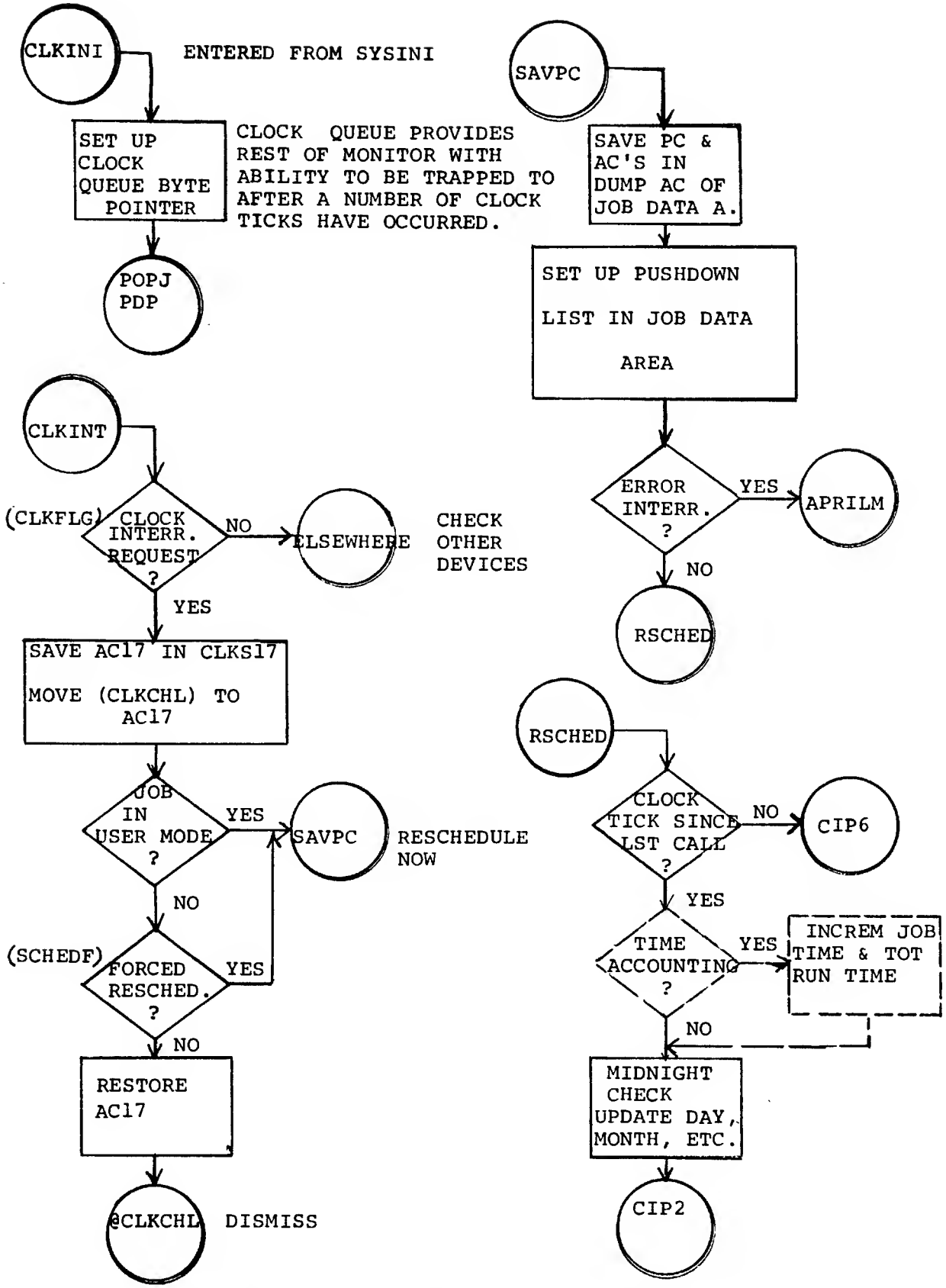



Figure 3. Flow Chart of Clock Interrupt Routine

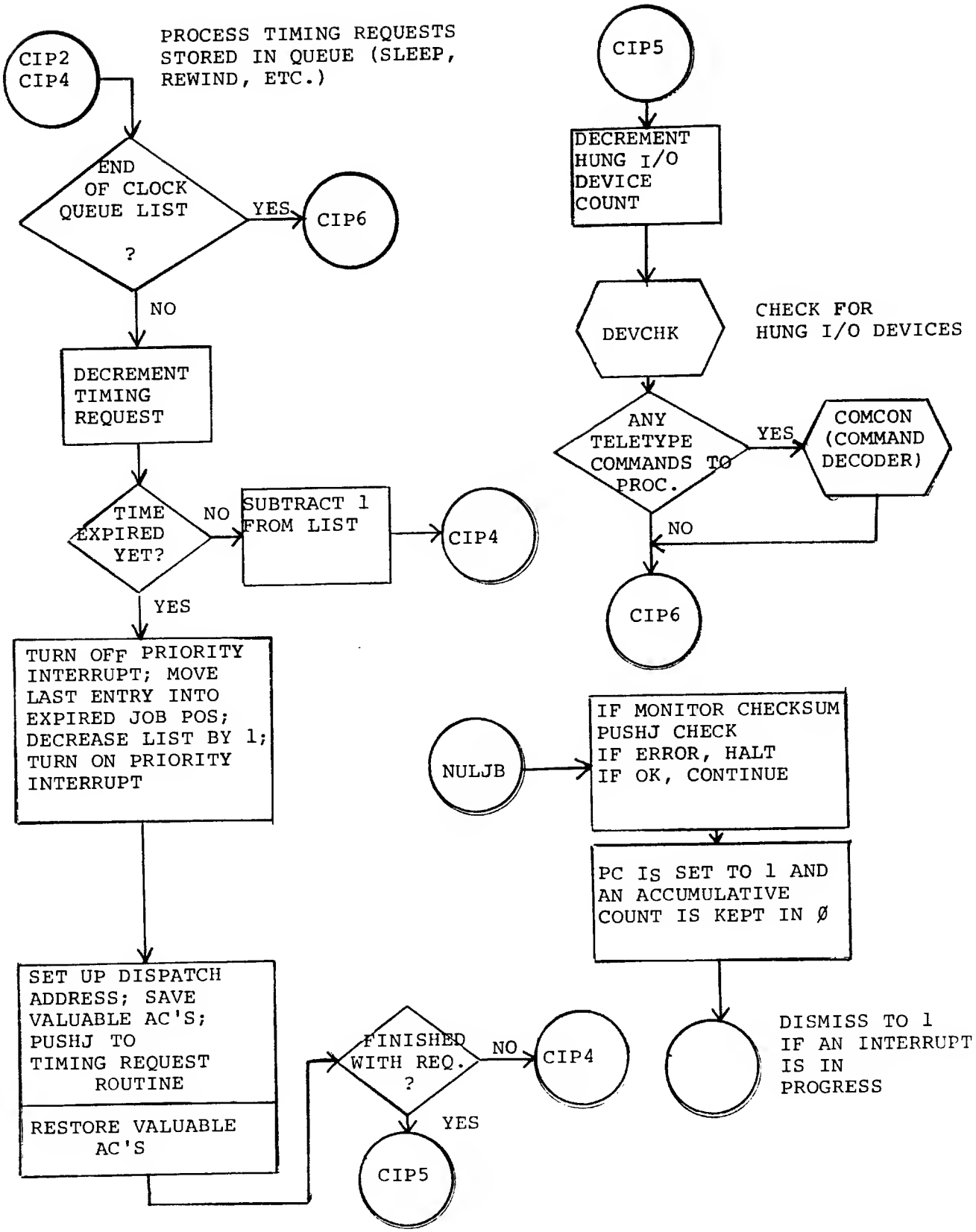



Figure 3 (Cont.) Flow Chart of Clock Interrupt Routine

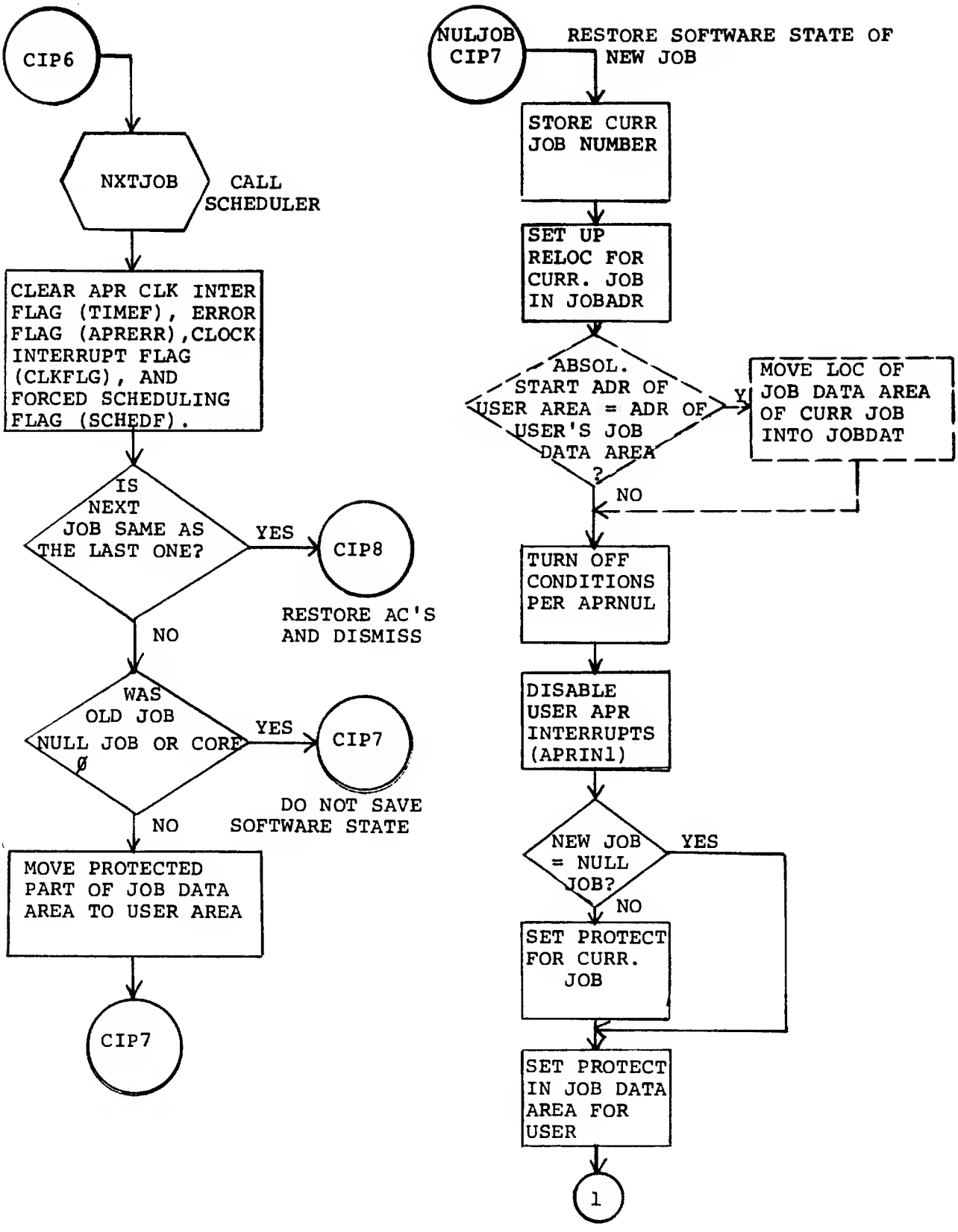



Figure 3 (Cont.) Flow Chart of Clock Interrupt Routine

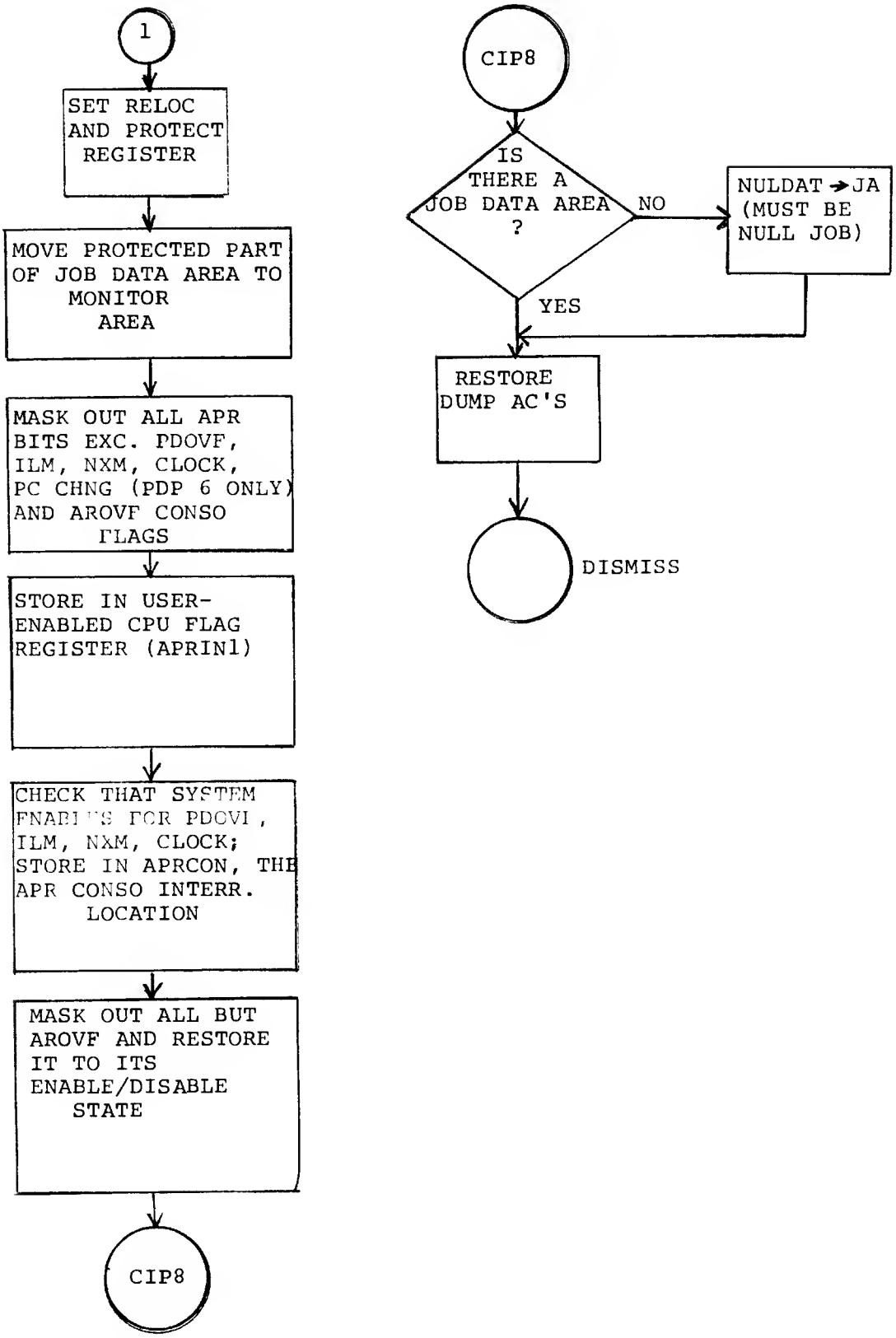



Figure 3 (Cont.) Flow Chart of Clock Interrupt Routine

Table 1

CROSS REFERENCE LISTING OF SYMBOLS IN APR AND CLK INTERRUPT ROUTINES[1]

| | |
|---|---|
| APRER | D2, F4 |
| ARRER2 | D2, F4 |
| APRER3 | D2, F4 |
| APRINT | D2, F3 |
| CIP2 | D6, F10 |
| CIP4 | F10 |
| CIP5 | D7, F10 |
| CIP6 | D7, F11 |
| CIP7 | D7, F11 |
| CIP8 | D7, F12 |
| CLKINI | F9 |
| CLKINT | D6, F11 |
| DEVCHK | F10 |
| NULJB | D8, F10 |
| NULJOB | F11 |
| NXTJOB | F11 |
| RSCHED | D6, F9 |
| SAVPC | D6, F9 |

[1] A D preceding a page number indicates that a description of the item is found on that page; an F indicates that the item appears in a flow chart on that page.

# 5

# PROGRAMMED OPERATOR SERVICE (UUOCON)

## I. DESCRIPTION

The function of UUOCON is to service in some manner those op codes which are trapped to absolute locations 40 and 41 by the processor hardware. These are op codes 000, 040 through 077, and (in user mode) 7xx (input/output), HALT (JRST 4, ), and JEN (JRST 10, ). In addition, the PDP-6 traps codes 001 through 037 as well.

The operations of UUOCON might, for the purpose of discussion, be divided into three sections.

1. Operator-independent preprocessing and dispatch;

2. Operator service (operator-dependent algorithms); and

3. Exit routines

Preprocessing includes saving of user accumulators if the machine was in user mode when the trap occurred (the Monitor may itself contain programmed operators), filtering out error codes, entering the user's UUO (User-Utilized Operations) handler if the machine is a PDP-6 and codes 001 through 037 are encountered, loading of accumulators with information to be used by the operator service routines, and dispatching to the proper service routine.

Operator service routines perform the algorithm designed for the particular UUO code, allowing the user to receive information about the system, to alter the operation of the system concerning his job, and to communicate with the input/output devices. A few specific examples are included in this chapter to demonstrate the information flow between the three sections of UUOCON and the user's job. Input/output UUO's are dealt with in the chapter on Input/Output Service.

The exit routines (normal or error) perform the setup necessary to return to the calling program or, in the case of errors, produce error messages and appropriately alter the status of the job. One important function of the normal exit routine is to check the status of the Scheduler before returning to the calling program. A software interlock between the Scheduler and UUOCON allows a UUO (which is, after all, one "instruction") to run to completion before the current job is stopped. The normal exit routine calls the Scheduler if the interlock flag was set sometime during the UUO processing.

## OPERATOR PREPROCESSING AND DISPATCH

### SPECIAL REGISTERS

A rather important function of this section is to place information about this user's job (i.e., the job that issued the UUO) into certain accumulators and index registers before dispatching. Therefore, these registers and their contents are described briefly before going into the operations of this section.

PDP A pushdown pointer to a 20-location list in the user's job data area. The first item placed in this list (JOBPDl) is the user's return, i.e., a copy of the PC word formed by the JSR in location 41.

PROG Contains a copy of the contents of JOBADR: XWD highest relative address, relocation for this job. Used as an index register by the system to relocate references to the user's program area.

JDAT Currently the same physical register as PROG, but, strictly speaking, contains the protection and relocation for references to the user's job data (JOBDAT) area.

UUO A copy of the programmed operator as trapped into location 40. The address PROG is set into the X field so that operator service can refer to (E) indirectly through UUO.

UCHN A copy of the AC address field of the UUO. UCHN stands for User Channel, which it is in the case of input/output operators.

DEVDAT[1] A copy of USRJDA (protected JOBJDA) for this software channel. This register contains Ø if this channel is unassigned. If the channel is in use, the left half of this word has status bits indicating what UUO's have been performed for the device so far; the right half contains the base address of the device data block (DDB).

IOS[1] A copy of the DEVIOS status word for the device on this channel.

DSER[1] A copy of the DEVSER word for the device on this channel. The left half of this word contains the address of the next DDB in a chain of all such blocks; the right half contains the base address of the dispatch table for this device's service routine.

---

[1] These registers are pertinent only to input/output programmed operators, but will be loaded, in any case, when an AC address (UCHN) happens to correspond to an assigned I/O channel.

FUNCTIONAL DESCRIPTION

The following is a narrative of the operator-independent preprocessing and dispatch section of UUOCON.

UUO1 — The user mode flag bit of the trapped PC word is used to detect whether the call is from the Monitor (as in a GET command) or from the user. If from the Monitor, certain AC's have been set up and a portion of the UUOCON coding can be skipped; control goes to UUOSY1. If the call is from the user and in the range 001 through 037 (PDP-6 only), then a software trap to the user's UUO handler is created, provided that the user has a nonzero address in his JOB41. If that location contains either Ø or an illegal address, an appropriate error message is typed on the user's Teletype and the job is stopped. If the call is from the user and is not in the 001 through 037 range, control goes to UUOSYS.

UUOSYS — The user's AC's are saved in the JOBAC part of his job data area and the contents of PROG, JDAT, and PDP are established.

UUOSY1 — This routine PUSH's the PC word (return address) as the first entry on the list and then tests the UUO for legality, now trying to exclude a 000 op code.

ILEGAL — If the routine is entered at this location, UUOERR is called, which types a message "ILLEGAL UUO . . ." and stops the job.

If the routine is entered with a skip, it sets the contents of UUO for indexing by PROG and then checks the op code for a value greater than 100 (illegal at this point). If the value is not illegal, accumulator UCHN is set up. If there is a device on this channel, DEVDAT, IOS, and DSER are set up. If no device has been assigned to this channel coincident with this UUO's AC address, the routine NOCHAN is entered. Otherwise, if this UUO is indeed an I/O operator of op code 72 or greater, then routine DISP1 is entered. Routine DISPØ is entered directly for non-I/O UUO's or I/O UUO's between codes 55 and 71 if the channel is found to be assigned.

DISPØ, DISP2 — This coding obtains an address from a 2-address-per-word dispatch table, using the op code as an index. If this UUO was from user mode, the service routine is dispatched to by a PUSHJ which puts the address of the user exit routine on the list as it jumps. If it was from the Monitor, then the desired address is already on the list and is left undisturbed when dispatching to the service routine.

NOCHAN This routine calls DISPØ if the UUO was from the Monitor, or if it was from the user and is not an I/O operator. If the UUO is a CLOSE or RELEASE operator, the successful return exit is called. Otherwise, the routine IOIERR is entered to type the message "I/O TO UNASSIGNED CHANNEL. . ." and stop the job.

DISP1 This routine "fakes" a successful return to the user if the UUO was a "long dispatch" one and the device service routine does not have a long dispatch table (this is an important concept in making user programs "device independent"; e.g., it enables a LOOKUP to a physical paper tape reader to be "successful"). If the device service routine is capable of performing long UUO's, the dispatch routine DISPØ is called.

## OPERATOR SERVICE

Before discussing a particular operator, let us first see how communication between the user's program and the operator service routine is made possible by setting up the AC's before dispatching. A most important point to note is that any Exec level software that refers to addresses in the user area must provide address checking equivalent to that performed by the hardware in user mode. A reference, especially one that stores information, must address a location equal to or greater than $(PROG)_{RH}$ and equal to or less than $(USRREL)_{RH}$. There are also some locations in the job data area which should be protected. Three address checking routines exist in Monitor and can be called from a UUO service routine.

UADCK1 This routine is called with a PUSHJ after loading AC1 with the address to be checked. It returns if this relative address is in the user's accumulator area or between JOBPFI (the top of the protected area of JOBDAT) and $(USRREL)_{RH}$.

UADRCK This routine is called in the same manner as UADRCK1, but considers accumulator area references illegal. Both UADRCK1 and this routine stop the job and print the message "ADDRESS CHECK . . ." message if a failure occurs.

IADRCK This routine, more forgiving than either of the above, is called (PUSHJ) with the address to be checked previously placed in TAC and PROG already set up. This routine considers an address acceptable if it lies between JOBPFI and the relative address in the left half of PROG. Failure is indicated by a no-skip return to the calling program, success by a skip return.

After careful address checking, access to user locations may be made in any of the following ways.

1. Fetch the contents of the effective address of the UUO.
   MOVE TAC, @UUO, where TAC is an accumulator available for use.

   NOTE

   Two things make "@UUO" work: (1) the hardware has computed the relative effective address at the time of the UUO trap, and (2) the UUO preprocessor routine has placed PROG in the index address field of AC UUO.

2. Store a result in the effective address location.
   MOVEM TAC, @UUO

3. Get an argument from the AC addressed by the UUO (recall that UCHN contains this AC address and that AC's are in the JOBAC area.

```
HRLI  UCHN, JDAT      ;relocate AC reference
MOVE  TAC, @UCHN      ;get contents
```

4. A routine STOTAC exists which stores the contents of accumulator TAC indirectly into the location addressed by UUO after checking the address (UADCK1 routine, and exits with a POPJ. To end a service routine by returning a result to the effective address of the UUO and immediately return to the user, the following instructions are executed.

```
MOVE TAC, result
JRST STOTAC
```

   If the call to STOTAC is made from the same level (with reference to the pushdown list) to which the preprocessor routine dispatched (via a PUSHJ), STOTAC's POPJ exit will return to the exit routine that followed the dispatch coding.

In returning to the user, one may wish to skip one or more arguments that followed the UUO, or to give a skip or no-skip return to signify success or failure of the operation. The UUOCON exit routine is designed to pass on to the user either a skip or no-skip return. If, when at the level equal to that following the dispatch, a POPJ PDP is used to exit, the user will receive a no-skip return. If the sequence

```
AOS (PDP)
POPJ PDP,
```

is used, a skip return occurs. This could be used to bypass one argument following the UUO (a system routine, CPOPJ1 performs this action if called by a JRST CPOPJ1). If it is necessary to bump up the user's return by more than one, the routine must take care of adding the correct quantity to the correct entry on the pushdown list (recall that, if the original UUO was issued by the Monitor, the preprocessor dispatch was <u>not</u> a PUSHJ). If, for example, two arguments are to be skipped in return to a user mode call, this sequence could be used.

```
AOS  -1(PDP)
JRST CPOPJ1
```

To give the same return to a call from the Monitor,

```
AOS  (PDP)
JRST CPOPJ1
```

Example

Presently, all operators that do not deal with some phase of input/output appear as subfunctions of the CALL programmed operator. To keep this example reasonably simple, we will choose one of these:

```
CALL  AC, [SIXBIT/RUNTIM/]
```

The referenced AC is loaded with a job number before the CALL, and the CALL returns the total running time (in "jiffies") of that job in the same AC.

The preprocessor routine of UUOCON sets up the standard accumulators and, using the UUO op code (CALL = 040), dispatches to UCALL. UCALL picks up the contents of the UUO effective address, the literal value RUNTIM. This argument is used to effect another dispatch to the routine JOBTIM, which gets the appropriate run time and stores it in the user accumulator. Before this second dispatch, the UCALL routine places the contents of the user's accumulator into TAC and changes the right half of UUO to contain the address of this accumulator. The accumulator ITEM is loaded with the job number of the currently running job.

When entered, the JOBTIM routine checks the contents of TAC for a valid job number and then uses it as an index to fetch from the TTIME table (where running times for all jobs are kept) the desired time and place it into TAC. A JRST STOTAC causes this result to be stored in the user's accumulator, now addressed by UUO, and return to the UUOCON exit routine.

## EXIT ROUTINES

### ERROR EXITS

Error exits, which do not allow a return to the user, occur when a UUO op code is illegal or an address supplied by the user is illegal. A nonimplemented UUO in the range 40 through 77, or a UUO of Ø, will stop the job with the error bit on (cannot continue) and print "ILLEGAL UUO at USER loc". An illegal op code (e.g., a DATAI in user mode) causes the job to be stopped with the error bit set and the message "ILL. INST. AT ...." to be printed. The HALT instruction stops the job, types "HALT AT USER loc.", but does not set the error bit. Thus, the CONT(INUE) command does function after a HALT.

When an illegal address is detected by a non-I/O UUO, the UUOERR routine is called to print the message noted above ("ILLEGAL UUO AT USER loc") and puts the job into an error stop. When a UUO is associated with a particular device, ADRERR may be called. ADRERR prints "ADDRESS CHECK FOR DEVICE dev: EXEC CALLED FROM loc", and results in an error stop condition.

NORMAL EXITS

If the original UUO was issued by the Monitor, the preprocessor dispatch was by a JRST rather than a by a PUSHJ. The service routine's last POPJ would bypass the user exit routine and go directly back to the Monitor coding following the call.

If the UUO was from the user, the service routine's terminating POPJ returns to location USRXT1 -1 (no-skip return) or a JRST CPOPJ1 returns to USRXT1, which passes a skip return to the user by adding 1 to the address on the pushdown list.

USRXIT This routine checks to see if the user has typed a CTRL C (↑C), or if the clock has ticked (software interlock), or if the system wants to stop this job (to swap it, for instance). If none of these conditions exists, the user's accumulators are restored and control is returned to his program. Otherwise, the Scheduler is called (USCHED) to take appropriate action. If the user's job continues in the future, control will come back here to restore the user's accumulators and continue the job.

## II. ADDING A PROGRAMMED OPERATOR

There are two ways to add a new UUO function to the Monitor. One is to use a previously unused op code (42 through 46 are open at the time of this writing - May, 1968). The other is to add a subfunction to the CALL operator. Before adding anything to any section of the Monitor, it is, of course, desirable to understand what is already there. Assuming that one already has this understanding and has written a tightly coded new routine that obeys the rules of address protection and uses as much existing coding as possible, we can investigate the process of getting this routine included in a running Monitor.

### ADDING A NEW OPERATOR

1. Edit the new coding into the source file for APRSER. If it is desired to make this routine a conditional feature, it may be enclosed in conditional assembly brackets preceded by a symbol like the feature test switches presently in use.

2. Edit into the UUO dispatch table, UUOTAB, the address of this routine in the proper half of the XWD found there. For instance, if you are adding a routine, UDUMP, as op code 43, you would replace XWD UUO42, UUO43 with XWD UUO42, UDUMP. Conditional assembly could be used

to set up the dispatch table entry if conditional assembly was used with the routine itself. For example,

| Routine Coding | Dispatch Table Entry |
|---|---|
| IFN FTDMPU, <UDUMP: . . . | IFN FTDMPU,< |
| (coding) | XWD UUO42, UDUMP |
| | > |
| > | IFE FTDMPU,< |
| | XWD UUO42, UUO43 |
| | > |

In this example, the routine will be assembled and the address of UDUMP is added to the dispatch table if the feature switch FTDMPU is nonzero.

3. In preparation for assembling the new APRSER, edit the correct feature test switch settings into the S (system parameter) source file, including any new ones you have established.

4. Assemble, naming as input first the S file, then the new APRSER file.

5. Use FUDGE2 to Replace the old version of APRSER with the new one in the file (SYS40 or SYS50) to be used in building your system.

6. Run System Builder, using the new file to build your Monitor. Follow the Build operating procedures.[1]

## ADDING A NEW CALL SUBFUNCTION

This method is an attractive alternative to adding an entire new operator when some job-number-dependent function is to be performed or when arguments to be passed are few. Recall that, before the CALL dispatches to a subfunction, it places the job number in accumulator ITEM, the contents of the UUO AC into TAC, and the address of the UUO AC into UUO, which has previously set for relocation. Thus, arguments or argument addresses can easily be passed via this accumulator. The CALL operator dispatches to a subfunction by searching a table of 6-bit names (UCLTAB) for a match with the contents of the UUO effective address and then selecting a corresponding jump address from a half word in a second table (UCLJMP). Alternately, the user may use the CALLI (CALL Immediate) operator and directly supply the index to the jump table. Because of the latter, any additions to the CALL dispatch tables must be appended to those entries already in existence.

### Example

The PDP-10 hardware will display a word in the console data lights when the instruction

```
DATAO  PI, [display information]
```

is executed. Let us add a new CALL to allow any user program logged in

[1] System Builder operating instructions can be obtained by listing the source file MONITR.OPR, located on the first of the three Monitor source tapes.

under project number 2 to display information by loading the data into AC and issuing the command

```
CALL  AC, [SIXBIT/CONLIT/]
```

Let us further specify that, if the user is not logged in with the proper project number (2), the call is to be treated as a no-operation. Finally, let us write the code in such a way that, in a Monitor with no login feature (feature switch FTLOGIN = Ø), this operator always works.

```
LIGHTS:

IFN FTLOGIN,<
     HLRZ   TAC1, PRJPRG (ITEM)          ;get project number
     CAIN   TAC1, 2                      ;equal to 2?
     >
     DATAO  PI, TAC                      ;display contents of AC
     POPJ   PDP,
```

After editing this coding into an appropriate area of UUOCON, the dispatch tables must be updated. This is done by adding one entry to the list following the NAMES macro which is called to build the two tables. An entry has the general form

```
X  function-name,  routine-address;  comment
```

To add our new display function, insert after the last name and before the LIST statement

```
X  CONLIT, LIGHTS; DISPLAY (AC) IN DATA LIGHTS
```

To create a working Monitor, follow steps 3 through 6 as outlined under "Adding a New Operator."

# 6

# SYSTEM INITIALIZATION AND RESTARTS

Once the Monitor has been loaded, the system is begun at Starting Address 140 absolute. The system immediately calls various sections of the subprograms FIRST, SYSINI, ONCE, and other Monitor subprograms. These subprograms elicit time and date information from the operator, establish PI channel trap locations, reset I/O devices, and initiate the null job (NULJOB) to make the system ready for use. The operator may treat any system malfunctions after this point in a variety of ways, depending on their severity. A System Dispatch Table, in the lowest section of the Monitor, provides the selection of a number of restart procedures, ranging from Executive Debugging (EDDT) in the case of mild system trouble-shooting, to complete Monitor reloading in event of catastrophic failure.

## FIRST

FIRST is the lowest Monitor subprogram in core and occupies 146 locations, beginning at location 140 absolute. The first part of this subprogram contains the System Dispatch Table, which is listed below.

Table 1

System Dispatch Table

| | | | |
|---|---|---|---|
| SYSDSP: | 140 (abs.) | JRST SYSINI | Monitor startup. Replaced by JRST IOGO on first pass of SYSINI. |
| | 141 | JRST DDTX | Run Executive DDT, if loaded. |
| | 142 | JRST SYSMAK | Make Job #1 the new Monitor. |
| | 143 | JRST SYSINI | Restart call to SYSINI. |
| | 144 | JEN NULJB1 | Run the null job (NULJOB). |
| | 145 | JSR ONCE | Operator dialogue at ONCE. |
| | 146 | JRST JSR2 | Call SYSINI, but bypass operator dialogue of ONCE. |
| | 147 | JRST SYSTOP | Write cut Storage Allocation Table onto disk and halt. |

In using this table, the operator starts up the Monitor from absolute location 140 and returns to other entries in the table in case of failure. In addition to system restarts, the operator can choose to run EDDT from this table for debugging any area of code in core (including the Monitor itself), or he can call SYSMAK, a subprogram which block transfers Job #1 to overlay the present Monitor. In the case of severe error conditions, the operator may choose to write out sections of the disk Storage Allocation Table currently in core onto the disk; absolute location 147 in the System Dispatch Table is used for this purpose, dispatching to SYSTOP (within FIRST), which immediately transfers to DSKSER (the disk service routine) to perform the writeout. In this case, the machine is finally halted at absolute location 20, which is the starting address to read in the routine DECDMP from the paper tape reader to be used in reloading the Monitor. Provided the subprogram ONCE remains undisturbed in core, as in the case where only debugging and patching have been performed on the Monitor subsequent to loading, the operator can restart the system from absolute location 145 adn enter the ONCE dialogue again to update system name, date, and time.

Apart from these procedures, FIRST simply contains locations that are set up and used to hold miscellaneous data required by the Monitor in its operation. Sections of the job data area (JDA) for the current job being run are copied into locations within FIRST for easy reference by the Monitor. This data may be updated by the Monitor and, whenever a new job is selected to replace the currently running job, updated information is read back into corresponding locations of the current job's job data area and the job is preserved until it is ready to run once more. By this process of copying the contents of the job data area, sections of the user's job data area are protected, even when the original copy within the user's area is inadvertently destroyed.

Monitor data areas within FIRST are set to Ø by SYSINI during initialization and most restarts.

Also, various system parameters are stored and defined within FIRST, although their values may be altered by the user at System Build time. These parameters have the following default values in absence of a request to the contrary.

Table 2

System Parameter Values Stored in FIRST

| Parameter | Default Value | Meaning |
|---|---|---|
| STDENS | 556 bpi with odd parity | Indicates density and parity of magnetic tape |
| DTTRY | 60 | Indicates number of DECtape rereads to be performed on read errors before issuing error message |

Table 2 (Cont.)

System Parameter Values Stored in FIRST

| Parameter | Default Value | Meaning |
|---|---|---|
| JIFSEC | $60_{10}$ | Number of clock ticks per second (power line frequency) |
| MTSIZ | $128_{10}$ words | Size of magnetic tape buffer |
| LPTSIZ | $24_{10}$ words | Size of line printer buffer |
| BLKQNT | $50_{10}$ | Furthest distance (in consecutive blocks) down a DECtape that a job can refer to without being rescheduled |
| NSPMEM | $2000_{10}$ | Number of nanoseconds per memory cycle (μsec. mem. speed) x 1000 |
| DETDDB | 0 | Number of extra Device Data Blocks available to attached and detached jobs |

Subprogram FIRST also contains 2- and 3-instruction special-purpose restore, restore and skip, skip and double skip return routines, used frequently throughout the Monitor to return from subroutine calls. These routines are labelled TPOPJ, TPOPJ1, CPOPJ1, and CPOPJ2, respectively.

## SYSINI

Subprogram SYSINI is called at Monitor startup time to perform the following tasks.

1. Initialize Monitor data areas by setting them to Ø.
2. Determine current core memory size and the area available to users.
3. Clear I/O devices, PI system, and APR of previous status settings.
4. Clear software flags and make other preparations for I/O transfers.
5. Run the null job (NULJOB).

Later, at system restarts, the user can dispatch to three locations within SYSINI (SYSINI, JSR2, and IOGO) from the System Dispatch Table (in FIRST).

The first operation of SYSINI is to clear all I/O devices as accomplished by a CONO APR, bit 19. Control then jumps to subprogram ONCE for dialogue with the user to obtain

1. Today's date and time;

2. Whether or not Executive DDT and SYSMAK are required for use;

3. Name of operator's Console Teletype; and

4. Whether or not the disk is to be refreshed.

SYSINI also calls LINKSR (in ONCE) to move the Executive DDT symbol table (if loaded at Build time) to upper core, leaving $700_8$ locations for dump routines TENDMP or DECDMP at the extreme top. This table is not protected, however, and may be lost later if user demands for core total up to all that remains after Monitor. Before return to SYSINI, system UUO (User-Utilized Operation) and PI channel trap locations 40 through $61_8$ (absolute) are set up. By overlaying location LINKSR +1 by itself with a return jump to JSR2 +1, LINKSR coding effectively disappears after first execution.

JSR2: SYSINI then proceeds to clear data and table areas of the Monitor, specifically the miscellaneous data locations of subprogram FIRST, and the Job Status, Protection-Relocation, Project-Programmer, Runtime, Clock Request, Teletype, and Pseudo-Teletype tables. Following this operation, each Device Data Block is cleared, in turn, of its logical name assignment and the DEVMOD flags:

Directory in Core
Teletype Attached
Teletype in Use
Assigned by Console
Assigned by Program

To determine the current size of core memory and the amount of core area available to users, SYSINI constructs a 10-word table (CORTAB) in which consecutive 1-bit bytes correspond to consecutive 1K blocks of core memory. The first location of each 1K block is refered to by the processor and the associated bit in CORTAB is set to 1 if either (a) the block is occupied by Monitor (locations $\leq$ SYSSIZ), or (b) the block is nonexistent (NXM flag set). Thus, at the end of this operation, 0 bits in CORTAB correspond to 1K blocks available to the user. At this point, for disk systems only, routine ACCINI (in DSKSER) is called to reserve the first virgin 1K block above Monitor (ignoring subprogram ONCE). This block, together with a remainder from the block in which Monitor itself terminates (again ignoring ONCE), serves as space to construct copies of a dummy disk block for each disk file opened, a well as to accomodate 4-word file-access tables read in at such times, and I/O buffering in case the users' areas are full. Accordingly, CORTAB is modified to indicate the removal of this additional 1K block from the user's pool.

IOGO: Although at Monitor startup time the first entry of the System Dispatch Table contains JRST SYSINI, this is overlayed by a JRST IOGO on the first pass through the subprogram SYSINI. Thus, at restarts from location 140, the ONCE subprogram dialogue, the CORTAB construction, and the disk preparations are bypassed. Also, device logical name assignments and DEVMOD flags are retained and control goes directly to IOGO, omitting the initial section of SYSINI.

IOGO turns off the PI system. Then, examining Device Data Blocks in turn, it clears all device-to-job-number assignments, except if assigned by user console command. Hung counts, I/O buffering, and device I/O status bits are destroyed. After this, a PUSHJ to NXTINI (in CLKCSS) cancels all shareable device wait queues and returns the devices to a common pool. REQTAB device request flags are -1 when no job is waiting or using a device, and AVALTB flags are Ø when devices are free with no jobs waiting.

At this point, SYSINI dispatches to device initialization routines (DINI) via the -1 entry of each device service dispatch table. Multiple devices, in fact, share the same dispatch table, and their controller requires initialization only once, not separately for each unit.

Jobs that have issued timing requests are cleared from the so-called clock request queue. During normal running, jobs in this queue have their preassigned task times decremented every clock tick. Here, however, CLKINI (in APRSER) is called and their times are immediately zeroed.

Before entering a loop to modify job status words, SYSINI takes time to clear location JOB, whose contents are the job number of the current job being run (providing it is running at UUO level). A mask is then set up, and (with the exception of Job #1) SYSINI enters a loop to clear each job status word (JBTSTS) of all bits except those corresponding to Job-Successfully-Logged-In (JLOG), Job-Number-Assigned (JNA), and Job-Swapped-Out-of-Core (SWP). Furthermore, for those jobs still in core, SYSINI does a PUSHJ to CLRJOB(in APRSER) to clear sections of their job data areas protected from I/O transfers (locations JOBPRT through JOBPFI). JOBDDT, however, which contains the starting address of the user's copy of DDT, is saved and restored during this operation. Other words to be cleared are JOBENB (APR-enable flags) and location JOBPD1 (PC word at user UUO's). Before returning to SYSINI, CLRJOB jumps to ESTOP (in APRSER) and flags the user to error stop (JERR bit of JBTSTS is set) so that he cannot continue.

If this is a swapping system, SYSINI calls QINI (in QCSS) to place all jobs on the null queue and, at the same time, to empty all other scheduling queues. In any case, SYSINI performs its remaining function of switching back on the priority interrupt system and starting the null job, which runs until the first user arrives at a console.

## ONCE

Subprogram ONCE is loaded as the highest Monitor routine in core. ONCE consists of the following four discrete sections.

1. LINKSR — SYSINI jumps to this location to execute a block transfer of the Exec DDT's symbol table (if loaded) to the top of core.
2. ONCE — Both location 145 in the System Dispatch Table and the subprogram SYSINI transfer here to enter dialogue with the user.
3. REFRESH — Called from the dialogue section (2) if the user requests that the disk be refreshed.
4. DFWUNS — Called from DSKINI, which in turn is called from SYSINI. Storage Allocation Table search entries are initialized in this section.

LINKSR — On entering this routine, the processor refers to successive 1K blocks of core until the NXM (Nonexistent Memory) flag becomes set, indicating that the top of core has been reached. Leaving $700_8$ locations at the extreme top of core for a dump routine (TENDMP or DECDMP), LINKSR proceeds to block transfer Exec DDT's symbol table (if the symbols were loaded at Build time) into the next-to-highest locations. DDTSYM (absolute location 36) is set up to contain the block pointer to this new position of the symbol table. Before returning to the calling program (SYSINI), LINKSR performs another block transfer to set up priority interrupt trap locations 40 through 61 absolute. These are established as follows.

Table 3

Priority Interrupt Trap Locations

```
4Ø (abs.)        Ø
UUOTRP: 41       JSR UUOØ
        42       JSR CH1
        43       JSP DAT, ERROR
        44       JSR CH2
        45       JSP DAT, ERROR
        46       JSR CH3
        47       JSP DAT, ERROR
        50       JSR CH4
        51       JSR DAT, ERROR
        52       JSR CH5
        53       JSP DAT, ERROR
        54       JSR CH6
```

Table 3 (Cont.)

Priority Interrupt Trap Locations

```
55      JSP DAT, ERROR
56      JSR CH7
57      JSP DAT, ERROR
60      Ø
61      JSP DAT, ERROR
```

Locations 60 and 61 are not yet fully implemented.

ONCE

This routine is dispatched to from location 145 of the System Dispatch Table and is also called by SYSINI (although bypassed on restarts from locations 143 and 146). ONCE enters a long dialogue with the operator; the dialogue sequence is as follows, beginning with a Monitor printout of

the name of the system

date of previous Monitor loading

and the message

MONITOR JUST LOADED

The Monitor then types

TYPE TODAY'S DATE AS ABOVE

and retrieves the operator's reply. If this reply is in the correct format (three 2-digit numbers, separated by hyphens and terminated by a carriage return; the digits in the form mm-dd-yy must also satisfy the requirements: $Ø < mm \leq 12$; $0 < dd \leq 31$; and $64 < yy \leq 99$), the date is converted by the formula

(yy-64)*12+mm-1)*31+dd-1

to represent the number of days since January 1, 1964, and stored in locations THSDAY and SYSDAT (in FIRST). If the reply is incorrect, the Monitor repeats the request.

On successful entry of the date, the Monitor types

TYPE A 4-DIGIT TIME

Again, the form of the reply is checked and, if acceptable, it is converted by the formula

$(h_1*10+h_2)*60+m_1m_2)*60*60$

an stored in location TIME (in FIRST) as the number of jiffies (clock ticks) past midnight. The first two digits, $h_1h_2$ are the hour, and the last two digits, $m_1m_2$, are minutes, and must fulfill the requirements:

$$0 \leq h_1 \leq 2$$

$$0 \leq h_2 \leq 9$$

$$0 \leq m_1m_2 \leq 59$$

At this point, the operator may choose to bypass the remaining dialogue and return immediately to SYSINI. To do this, he simply types a carriage return immediately after the 4-digit time. Otherwise, if ALTMODE is typed, Monitor continues by printing a listing of how many units of each I/O device are present. To compile this list, the coding cycles through chained Device Data Blocks and compares physical names. The list appears as follows.

IO CONFIGURATION

$n_1$ DTA'S

$n_2$ MTA'S

$n_3$ CDR'S

.

.

.

$n_m$ TTY'S

Following this, another request is typed to the operator.

TYPE OPERATOR'S CONSOLE DEVICE NAME (CR IF NONE)

The name entered is stored in DEVOPR (within FIRST), provided it is acceptable, and will be used synonymously with the device name OPR.

To aid in the discussion of the remaining dialogue, a map of the upper section of Monitor as it appears in core is given below.

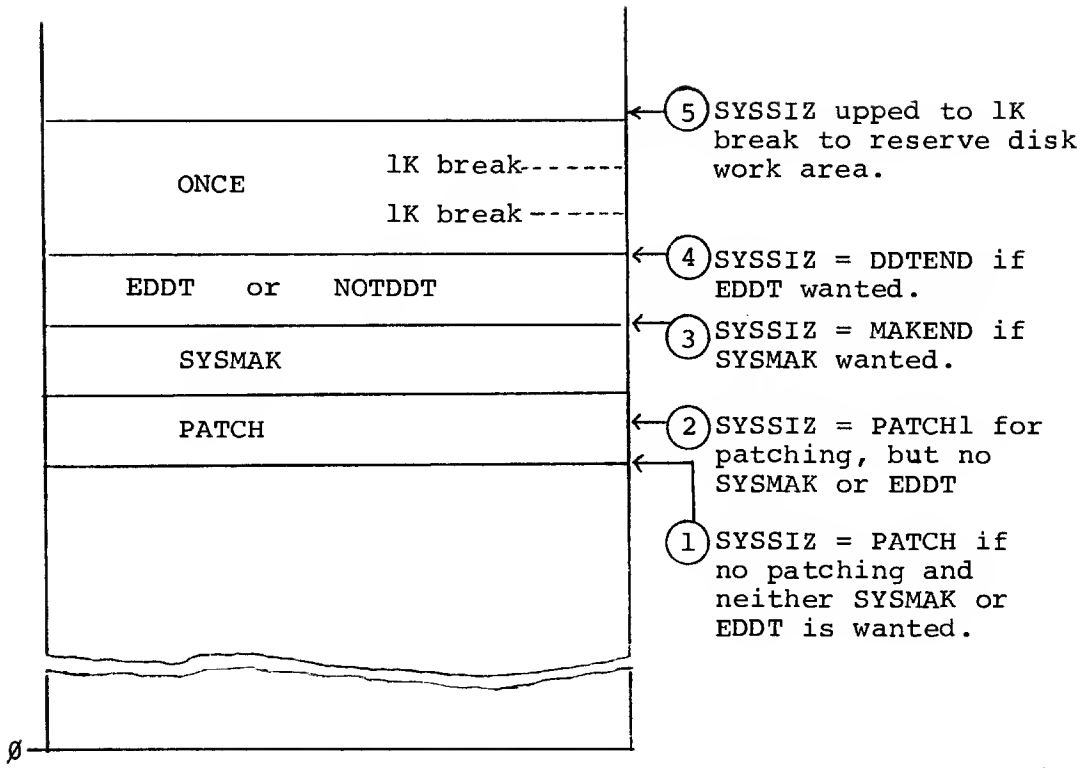



Figure 1. Map of Upper Section of Monitor

The user has already chosen at System Build time whether or not Executive DDT is to be loaded. Now he must choose whether or not to make use of EDDT or of SYSMAK (a routine used for overlaying the lower section of core with a new Monitor built or loaded into Job #1's core area.

Location SYSSIZ contains a value that is taken to be the highest address of the Monitor (ignoring subprogram ONCE) and above which all core is available to users. Its value is changed several times during the process of preparing Monitor for startup. At this point in the ONCE routine, SYSSIZ is set to the lowest location of PATCH (#1 in Figure 1).

Although $500_8$ locations are reserved for Monitor modifications by the PATCH subprogram, it is assumed that when a Monitor is first received from Digital by a customer, there are no patches present. Thus, the pointer SYSSIZ is positioned to point to the lowest location of PATCH so that, if the user should choose not to load or run EDDT or SYSMAK, the area occupied by PATCH would be available to users. However, in the event that a user wishes to make his own modifications to the Monitor, PATCH would be used to hold the new sections added and SYSSIZ would not contain the lowest address of PATCH, but the first address above the inserted patches.

In this way, the patches are subsequently protected from the users, as are all other parts of the Monitor (except ONCE). The user must make this adjustment to SYSSIZ himself at the time of patching. He does this by replacing the contents of location ONCE + 1 (in subprogram ONCE) with the code

```
MOVEI TAC, PATCH1
```

where PATCH1 (see #2 in Figure 1) is assumed to be a label assigned to the highest location occupied by the inserted patches.

Continuing with the dialogue, Monitor asks

```
DO YOU WANT SYSMAK?  (TYPE Y IF YES, CR IF NO)
```

If the user's reply is affirmative, location SYSSIZ is set to the highest location of SYSMAK (see #3, Figure 1), this being treated as the new upper limit of Monitor (ignoring ONCE) after having taken into account the extra core required to include SYSMAK. Otherwise, if the reply is negative, SYSSIZ remains set to its present value.

Next, the Monitor asks

```
EXEC DDT?
```

If EDDT is to be used and this program has been loaded at System Build time, the user types Y and the contents of SYSSIZ are once more incremented, this time to point to the last location of EDDT.

As mentioned under SYSINI, the disk service routine DSKSER reserves an additional 1K or more core locations immediately above the final setting of SYSSIZ (as determined above), and these locations lie within the subprogram ONCE. Thus, after users have begun to use the system, subprogram ONCE cannot be expected to remain in usable shape. In fact, the area occupied by ONCE will be claimed as user area, and over-written as disk buffer and file access table copying space. Indeed, restarts from absolute location 145 are ineffective after any users have been on the system, since dispatch is made to ONCE, a subprogram which may no longer exist in core. Absolute location 145 is primarily reserved for restarting the Monitor after patching, or perhaps debugging, have been the only operations performed following Monitor loading.

REFRESH

Finally, if this is a disk system, the ONCE code jumps into a dialogue to discern whether or not the disk is to be refreshed. Return from this diversion is immediately to the calling program (either SYSINI or absolute location 145 of the System Dispatch Table).

| | |
|---|---|
| DFWUNS | Initializes Storage Allocation Table search entries. This routine is called from DSKINI, which in turn is called from SYSINI. |

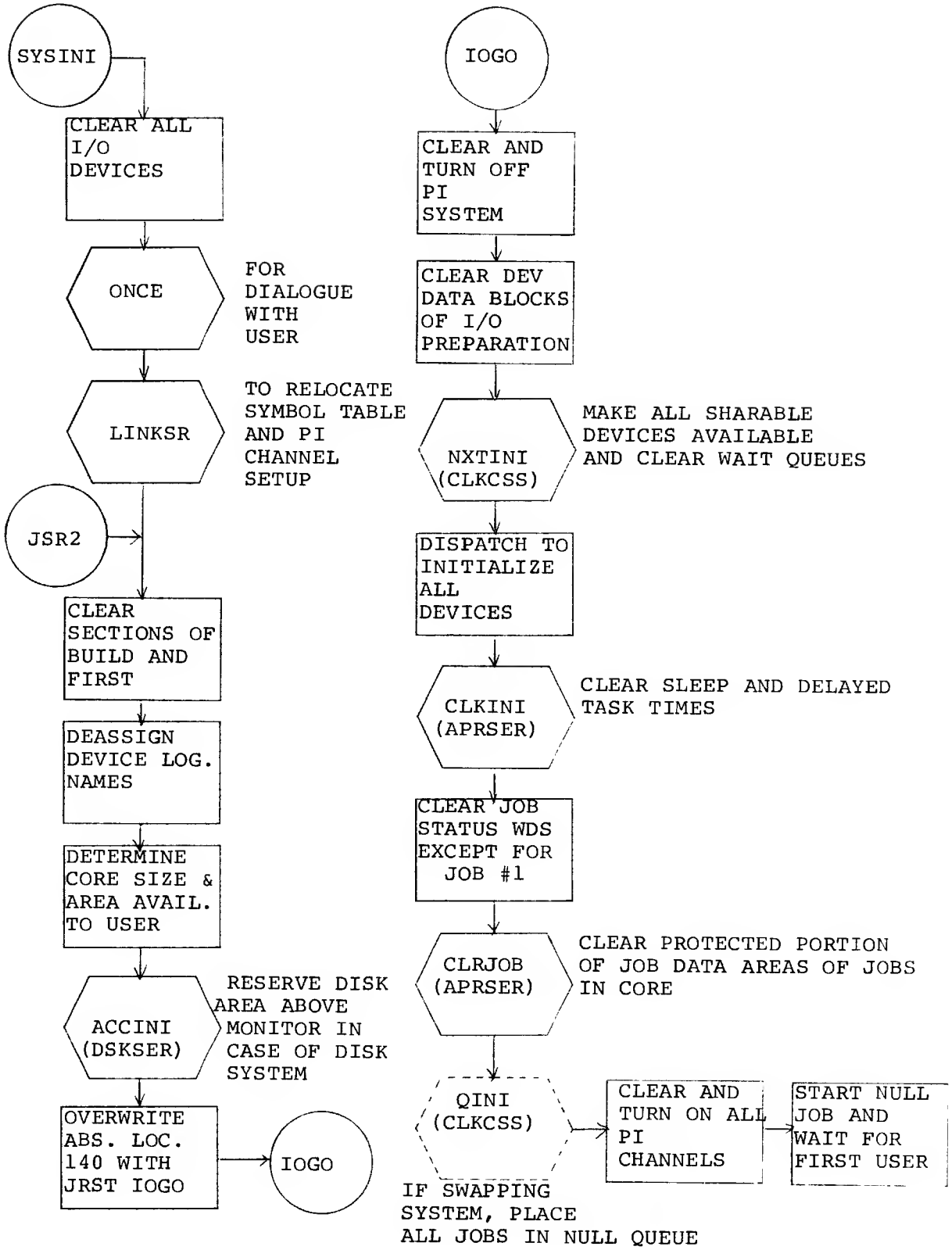



Figure 2. System Initialization Flow Chart (SYSINI)

Table 4

Cross-Reference Listing of System Initialization Symbols[1]

| | |
|---|---|
| ACCINI | D4, F12 |
| BLKQNT | D3 |
| CLKINI | D5, F12 |
| CLRJOB | D5, F12 |
| CORTAB | D4 |
| CPOPJ1 | D3 |
| CPOPJ2 | D3 |
| DEVOPR | D8 |
| DFWUNS | D6, D11 |
| DSKSER | D2 |
| DTTRY | D2 |
| EDDT | D2, D9 |
| ESTOP | D5 |
| FIRST | D1 |
| IOGO | D5, F12 |
| JIFSEC | D3 |
| JSR2 | D4, F12 |
| LINKSR | D4, D6, F12 |
| LPTSIZ | D3 |
| MTSIZ | D3 |
| NSPMEM | D3 |
| NXTINI | D5, F12 |
| ONCE | D6, D7, F12 |
| PATCH | D9 |
| QINI | D5, F12 |
| REFRESH | D6, D10 |
| STDENS | D2 |
| SYSDAT | D7 |
| SYSDSP | D1 |
| SYSINI | D3, F12 |
| SYSMAK | D2, D9 |
| SYSSIZ | D9 |
| SYSTOP | D2 |
| THSDAY | D7 |
| TIME | D8 |
| TPOPJ | D3 |
| TPOPJ1 | D3 |
| UUOTRP | D6 |

[1] A D preceding a page number indicates that a description of the item is found on that page; an F indicates that the item appears in a flow chart on that page.

# 7

# CONTEXT SWITCHING

Context switching (that is, setting up to run the next job) is handled by the subroutine CLOCK, which is in APRSER. The hardware state of the machine is saved before the Scheduler is called at location CIP6. If the Scheduler determines that a new job is to be run, the software state of the old job is saved (CIP6+7) and the software and hardware states (CIP7 and CIP8, respectively) of the new job are restored. The hardware state of the machine is stored in what are called the dump AC's (locations 20 through 37) in the job data area (locations 0 through 137) for the particular job. The software state of the machine is stored in the job data area locations JOBENB (43), JOBCHU (72), JOBPC (73), JOBDDT (74), and JOBJDA (75 through 114).

CIP6+7 — The software state of the old job is saved at CIP6+7. This occurs after it has been determined that the next (new) job is different from the one which was running. If the next job is the same as the last, the hardware state of the machine is restored and the interrupt is dismissed (at CIP8). If the old job was the null job or core 0, control is transferred to CIP7 (alias NULJOB), and the software state of the old job is not saved.

The actual saving operation sets up a block transfer to move four words from the Monitor location USRPRT to the job data area JOBPRT. The ending of the block transfer may then be modified to save all software channels assigned. These software channel words start at the fourth word (channel Ø) of the block transfer and may add up to 17 words (for 17 channels) to the transfer.

CIP7 — This location is the beginning of the routine which restores the software state of the new job. This is also an entry point for SYSINI to start the null job. The protection and relocation of the new job are placed in JOBADR. All APR flags are cleared and the user APR interrupts are disabled. If the new job is the null job, it is set up to keep count in ACØ.

If the new job is not the null job, the software protection registers (USRREL and JOBREL) are set and the hardware protection and relocation register is set. The locations starting at JOBPRT are block transferred to USRPRT. As in the saving of these registers, this block transfer is modified to include all software channels assigned; any unassigned software channels between Ø and the highest assigned software channel will be included in this block transfer.

The word JOBENB (location 43) in the job data area stores

bits for user traps to the APR in the right half and PC change and overflow enable/disable bits in the left half. The bits in the left half are masked and stored in APRINI and APRCON (the user trap CONSO instruction and the APR CONSO interrupt location, respectively). The APRINI is masked to contain only flags for pushdown overflow, illegal memory, nonexistent memory, clock enable/disable, PC change, and arithmetic overflow bits. APRCON is masked to contain the above flags with the additional stipulation that the pushdown overflow, illegal memory reference, nonexistent memory, and clock flags must be set. The arithmetic overflow enable/disable bits are then masked out and set with a CONO to the APR.

CIP8 The is the starting location for the routine which restores the hardware state. If the job is not the null job, the AC's are restored from that job's dump AC's (locations 20 through 37). The clock interrupt is then dismissed and control is transferred to the address stored by the software restore in USRPC.

NOTE

A discussion of these routines, viewed in a slightly different perspective, can also be found in Internal Memorandum #4 of this series, entitled APR and Clock Interrupt Routines, Programming Department Memo # 100 150 004 00.

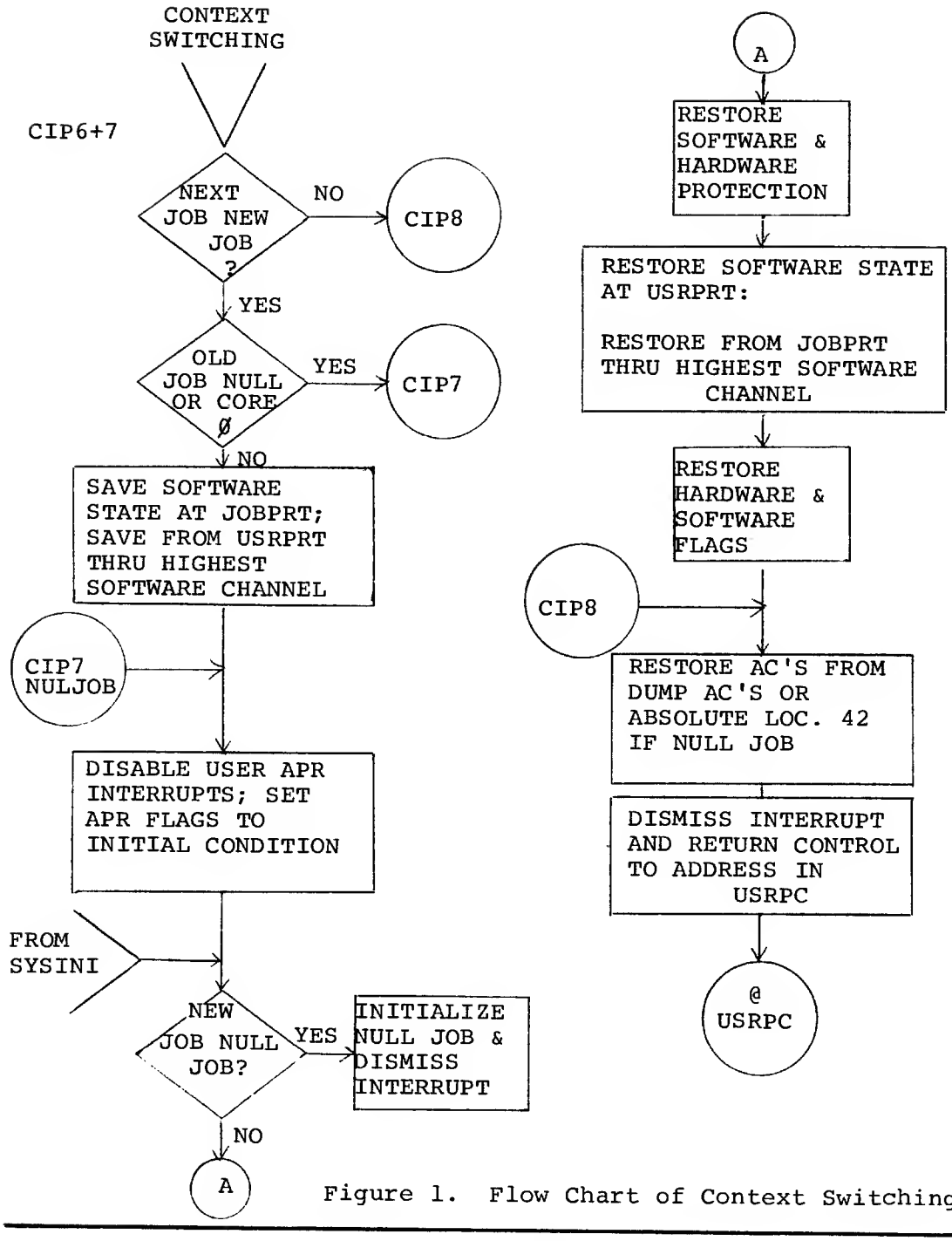



Figure 1. Flow Chart of Context Switching

Table 1

Cross Reference Listing of Context Switching Symbols

| | | | | | |
|---|---|---|---|---|---|
| APRCON | D2 | JOBADR | D1 | USRPC | D2, F3 |
| APRINI | D2 | JOBENB | D1 | USRPRT | D1, F3 |
| | | JOBPRT | D1 | USRREL | D1 |
| CIP6+7 | D1, F3 | JOBREL | D1 | | |
| CIP7 | D1, F3 | | | | |
| CIP8 | D2, F3 | NULJOB | D1, F3 | | |

8

# INPUT/OUTPUT PROGRAMMED OPERATORS AND DEVICE SERVICE ROUTINES

## I. SOFTWARE LINKS BETWEEN USER AND DEVICE

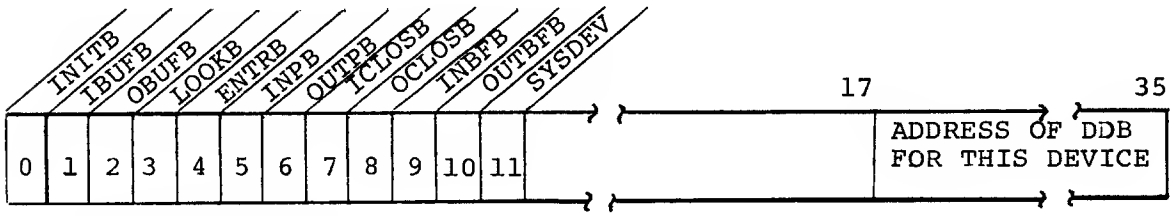
User input/output is made possible by the programmed operators and several tables existing in Monitor and the user's job data area. When desired, software linkage is made between a user program and a device (file) via these tables. For each physical device (or each active file, on disk) there is a device data block in Monitor describing the characteristics of the device: name, legal data modes, standard buffer size, and location of the service routine dispatch table. For a complete description of these tables, see Section III. When a device is assigned to the user and is being used by him, certain locations in the device data block (DDB) will contain certain information concerning the current activity: the job number using the device, status, data mode in which the file is to be read or written, and location of the user's buffers. The user, or some part of the Monitor, may look in the DDB to find out something about this device. The device service routine may obtain information about the user of this device by taking the job number from the DDB and referring to one of these Monitor tables indexed by job number (job status JBTSTS, job project-programmer PRJPRG, core assignment JBTADR, etc.).

The user's link to the DDB, and thus to the device, is one word in a 16-word table, JOBJDA, in his job data area. A location in this table is accessed by an index called the "software channel", supplied by the user. Figure 1 depicts one such location containing the address of the DDB and bits indicating the UUO operations done so far for this device. The user never directly accesses a JOBJDA location, but the Monitor does at UUO level using the software channel number specified in the AC field of the I/O operator. For protection, these locations are copied into the Monitor (starting at Monitor location USRJDA) when a job is running and Monitor works with these copies, restoring them to JOBJDA when the job is not running. A seventeenth location, JOBHCU (USRHCU), contains the channel number of the highest channel in use. These 17 locations, and others, are in an area of JOBDAT protected from input data transfers. The symbol JOBPFI ("protect-from-input") is a relative location in the job data area below which data must not be transferred, and user buffer addresses are checked against this value by those I/O UUO's concerned.

The last table to consider is the jump table or dispatch table in the device service routine. This table (see Figure 11) links the device dependent coding in the routine with the device independent portion of UUOCON. The address of this table is in the right half of the DEVSER word in the DDB. UUOCON dispatches to this address plus or minus an appropriate index so that the service routine may perform whatever is necessary to service this UUO (start or stop the device, initialize hardware or software registers, etc.). Much of the work of UUO service is

done by the device independent UUOCON routines and, even after dispatch to the device dependent routine, portions of the Monitor (IOCSS, in particular) are called as subroutines. The ultimate effect is that the user's program deals with all devices (or files) in a similar manner and the device service routine has only to interface some specific hardware device with the general coding of UUOCON. This latter topic, along with interrupt level operations, is dealt with in Section III, "Device Service Routines".

The next section describes the operations within UUOCON as it handles the communication between user and device.




UUO PROGRESS BITS

| | |
|---|---|
| INITB | INIT or OPEN has been performed |
| IBUFB | An input ring header was specified (by INIT) |
| OBUFB | An output ring header was specified (by INIT) |
| LOOKB | A LOOKUP has been performed |
| ENTRB | An ENTER has been performed |
| INPB | At least one INPUT has been performed |
| OUTPB | At least one OUTPUT has been performed |
| ICLOSB | A CLOSE input has been performed |
| OCLOSB | A CLOSE output has been performed |
| INBFB | An input buffer ring has been set up |
| OUTBFB | An output buffer ring has been set up |
| SYSDEV | This is the system tape device |

NOTE: This word is completely cleared by RESET or RELEASE UUO's

Figure 1. JOBJDA or USRJDA Word Contents

# II. I/O OPERATORS

## REVIEW OF USER I/O

This section assumes previous familiarity with user I/O programming as described in Chapters 4 and 5 of the PDP-10/40, PDP-10/50 Time-Sharing Monitors manual (DEC-10-MTC0-D).

Two methods are used to effect data transfers: unbuffered and buffered. In unbuffered modes, the user supplies to the device the address of a command list in his program area. This list consists essentially of block pointers to relative locations in the user area to or from which data is to be transferred. Upon initiating such a transfer, the user's job is scheduled into an I/O Wait where it remains until the device signals (to the Scheduler) the completion of the entire transfer. The device, at interrupt level, follows the command list in making the transfer until a termination word (null) is found and then notifies the Scheduler.

Buffered data transfers are made using a ring of buffers set up in the user area. A ring may contain one buffer or as many as will fit in the job area. A 3-word ring header in the user's program contains a byte pointer and item counter to be used by that program in accessing the "current" buffer (the one the user's program is working on). The device data block of the device involved in this data transfer contains like information concerning that buffer which is current to the interrupt level data transfers (see Figure 2). Monitor routines called by UUO's (INPUT or OUTPUT) work to supply a new buffer to the user, setting up the ring header appropriately. Monitor routines called at interrupt level likewise supply a new buffer for the device to work on, updating the pointer and item count in the DDB. To prevent the user and the device from using the same buffer at the same time, each buffer contains a use bit in the second word of the buffer header that is checked and altered by the Monitor's buffer handling routines. At UUO or interrupt level, a 1 means that the buffer is full and a 0 means that the buffer is not full. If the user "overtakes" the device and requires as his next buffer the one currently being used by that device, the user's job is scheduled into an I/O Wait. Upon completion of its use of that buffer, the device calls the Scheduler to reactivate the job. If the device "overtakes" the user, the device is stopped (always at the end of a buffer) and is restarted when the user finished with the buffer. (Input devices are not actually restarted until all but one of the buffers in the ring have been emptied by the user.)

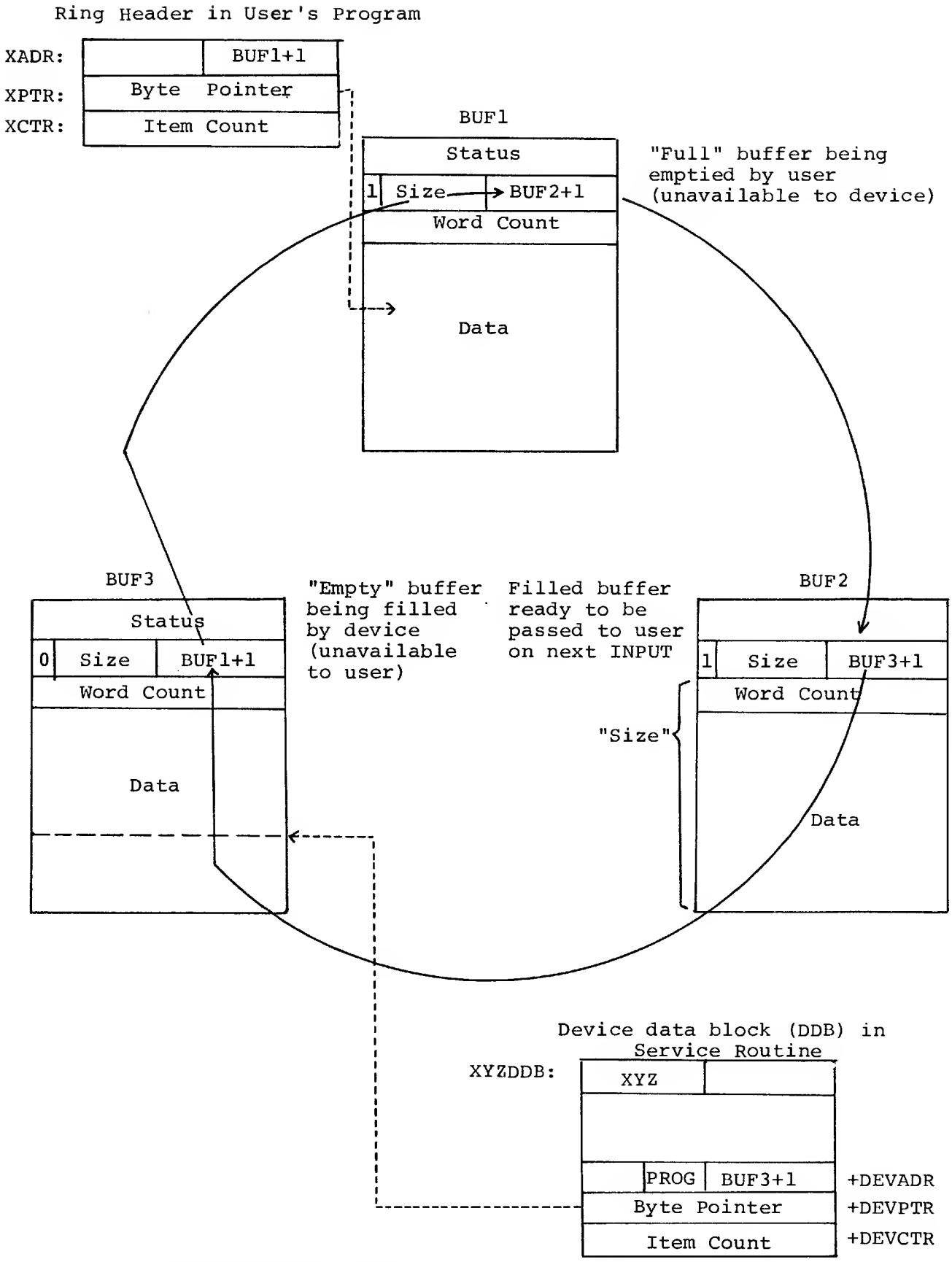



Figure 2. Buffered Data Transfer Between an Input Device and User Via a 3-Buffer Ring

## INIT AND OPEN OPERATORS

These operators assign a device to a user's program, establishing the link between the software channel and the device data block. The initial status, including data mode, is placed in the DEVIOS word, and the DEVBUF word is given the relative addresses of the output ring header and input ring header, if specified. A byte pointer size field according to mode is placed in the second word of each ring header. An error return to the user occurs if the device is not found or is unavailable at this time. No dispatch to the device service routine is necessary for INIT or OPEN. See Figure 3.

## INBUF AND OUTBUF OPERATORS

These operators create a buffer ring in free locations in the user area. The number of buffers is specified by the user as the effective address of the operator (one buffer is established if that value is equal to or less than 1). The size of each buffer data area is obtained from the righthand 12 bits of DEVCHR for the device assigned to the software channel. Two words are added to this amount for buffer head use.

As each buffer is appended to the ring, the last word of the buffer is address checked. A use bit of 0, the buffer size, and the link to the next buffer in the ring is inserted into the 2nd word of the new buffer. The 2nd word of the last buffer created is made to point to the first buffer, thereby closing the ring. JOBFF is then updated to point to the first location beyond the last buffer of the ring. Depending upon the operator, either DEVIAD or DEVOAD receives the relative address of the 2nd word of the first buffer. The first word of the user's ring header is then set with a "virgin" ring use bit and the address of the first buffer. No dispatch to the device service routine is necessary.

See Figure 4.

## INPUT OPERATOR

### Dump (Unbuffered) Mode

The device independent part of this operation is quite simple. The Monitor waits, if necessary, until the device is inactive, then dispatches to the device service dump mode input routine (device's dispatch table entry indexed by "DDI"). The device service routine takes care of command list checking, initializing its interrupt level program, and starting the device. Upon return to UUOCON, the routine WAIT1 is called to place the job in an I/O Wait until the device becomes inactive. Dump mode input, therefore, goes on at interrupt level for this job

while other user's jobs are running. When the device service routine recognizes the end of input activity, it calls a routine (SETIOD - "set I/O done") that notifies the Scheduler to take this job out of I/O Wait. The job (at the appropriate time) then commences running, in this case at the UUOCON normal exit.

## Buffered Modes

To somewhat simplify this description, let us take the first INPUT and subsequent INPUT's as two separate cases. Reference should be made to the flow chart (Figure 5).

Case 1 - First Issuance of INPUT operator

IN — If the device is doing output (IO=1), force output to stop at end of next buffer and wait until it has done so. Zero the input close bit in the left half of DEVDAT (a copy of USRJDA for this channel).

IN1 — If a buffer ring has not been established (by INBUF, for instance) or if this is the first INPUT ("virgin" ring), go to do first input, INPUTF.

INPUTF — Clear the header ring use bit. If a ring has not been set up, go to INPUT3. Otherwise, take the address of the first buffer from the first word of the ring header and place it in DEVIAD of the DDB. PUSHJ to CALIN to start the device. Device service then returns here and control falls into INPT0A.

INPT0A — Is the buffer's use bit a 1 yet? (Probably not, because we have just started the device.) If not a 1, call INPT2.

INPT2 — Calls WSYNC to place the job in an I/O Wait state until the device calls SETIOD at the end of the buffer. If the buffer use bit is now a 1, go to INPUT2.

INPUT2 — Gets the word count from the buffer and calls IOSETC which sets up the correct byte pointer and item counter in the user's ring header, and then exits to the user.

CALIN — If the device has previously sensed end of file (IOEND=1), return immediately; otherwise, address check the buffer limits and dispatch to the device's input routine. The device routine initializes itself for interrupt-level data transfers, starts the device, and returns. CALIN then returns to the calling routine.

Case 2 - Subsequent Issuances of INPUT Operator

IN1 — If a buffer ring exists and has been used, test the use bit of the buffer now being returned by the user. If the use bit is 0 (as when a user's program is doing OUTPUT from this same ring), bypass header updating and go to INPT1. If the use bit is a 1 (probably more typical), clear it and advance the header first word to point to the next buffer in the ring. If the device is active, (IOACT=1), go to INPT0C; else, determine if it is time to make the device active. For all devices except the Teletype, this determination is made by looking at the use bit of the buffer beyond the one to be returned to the user. If it is 0, CALIN is called to start the device. In the case of Teletype, the new buffer's use bit is examined becuase a Teletype has a Monitor buffer in addition to the user's ring. Whether or not a call to CALIN is made, control now goes to INPT0C.

INPT0C — The new buffer's use bit is fetched for examination, and control passes to INPT0A.

INPT0A — If the buffer's use bit is a 1, go to INPUT2, as described under Case 1; otherwise, go to INPT2.

INPT1 — If the device is active, go to INPT2; else, call CALIN to start the device and then return to INPT2.

INPT2 — Calls WSYNC to put the job in an I/O Wait if the device is active. Control returns if the device is not active or when the device calls SETIOD. If the buffer use bit is a 1 upon return, control goes to INPUT2; if not, control goes to INEOF.

INEOF — Did control get to here because of an error or end of file? If not, go to INEOFE. If so and the cause was end of file (IOEND=1), then set the user end file bit, IODEND. The use of these two end-file bits simplifies user programming by guaranteeing that when he detects end of file (with a STATX operation), there is no residue of information in a buffer. For instance, if a user's INPUT causes a buffer to partially fill and then an EOF to be detected by the device, the device returns the "full" buffer to the user while remembering the end condition (IOEND=1). IOEND is not detectable by the user, so he empties the buffer until the reduced header item count forces a call to INPUT. The IOEND bit prevents CALIN from starting the device which ultimately causes (at INPT2) an immediate return from WSYNC with a 0 buffer use bit. INEOF now finds IOEND=1 and turns on IODEND, then returns to the user.

INEOFE — Control should never get here. If it does get here and any of the job status STOPIO bits are on, control goes back to UUOCON dispatch and the input UUO is repeated to make the device do something (fill one buffer, or return with EOF or error bits). If no STOPIO bits are set, the error message routine UERROR is called to print

```
? ERROR IN MONITOR AT EXEC nnn
```

and stop the job.

## OUTPUT OPERATOR

### Dump (Unbuffered) Mode

UOUT — Set output UUO bit and clear output close bit in left half of DEVDAT.

OUT — If device is busy doing input, wait until the next bufferful. When stopped, if dump mode, go to OUTDMP.

OUTDMP — Call WSYNC to make sure device is inactive, then dispatch to the device's dump mode output routine (dispatch table "DDO" index). The device routine checks the command list, starts the device, and returns. WAIT1 is called to place the job in an I/O Wait state until the entire output is done. Control then returns to the calling routine.

### Buffered Modes

OUT — If not dump mode, call OUTA to get new buffer address if user specified one. If a buffer ring has not been set up or if this is the first OUTPUT, go to OUTF. Otherwise, if the user is not computing his own word count, take the header item count, convert it to word count, and store it in the third word of the buffer. Don't compute if the user so indicates. Go to OUT2.

OUT2 — Turn on the buffer's use bit, then advance the header to the next buffer. If the device is not now active, dispatch to the service routine to start it going. If the new buffer is not empty, call WSYNC to put the job in I/O Wait until the buffer is empty (the device calls SETIOD at interrupt level to take the job out of Wait). Go to OUTS.

OUTS — Calls BUFCLR to clear the buffer, then calls IOSETC to set the ring header byte pointer and item counter for this device. Return to calling routine.

OUTF — If a buffer ring has not been established, call UOUTBF (OUTBUF operator routine) to set up a 2-buffer ring. Go to OUTF1 in any case.

OUTF1 — Clear the ring use bit. Supply the address of the first buffer to DEVOAD of the DDB. Go to OUTS.

OUTA — If a new buffer address is not specified, return immediately. (NOTE: The mask used in making this test ignores bits 34 and 35. This is because OUT is called by the CLOSE routine in which case one of these bits may be a 1 to inhibit closing "half" the channel. We don't, in that case, want to believe that location 1 or 2 is being specified as the start of a new buffer!) If an address is specified, wait until the device is inactive, then put the new address into the user's ring header and DEVOAD. Mark the ring as being referenced (clear the use bit) and return.

N.B. Observe that if the first OUTPUT does specify a buffer address, the assumption is that the buffer contains data already, i.e., this will not be treated as a "dummy" output.

See Figure 6 for a flow chart of the OUTPUT operator.

## CLOSE OPERATOR

CLOSE1 — Calls WaIT1 to be sure that the device is inactive before proceeding. If an input close is requested and the file has previously been closed, go to UCLS2. Otherwise, if the file was read in DECtape save mode (2), return. If not save mode and not dump mode, go to UCLSBI to close buffered input. If dump mode, dispatch to the device service routine input close function (dispatch index "DCLI") and return to UCLS2.

UCLSBI — If an input buffer ring was never established, go to UCLS2; else, if this is a long dispatch table device, dispatch to the device service routine (dispatch index "DCLI"). Then get the address of the first buffer from the ring header. If 0, go to UCLS1; otherwise, go to UCLS0.

UCLS0 — Address each buffer in the ring, clearing its use bit. Go to UCLS1.

UCLS1 — Set the ring use bit to 1 ("never referenced")and clear the item count word to 0. Clear both end-file bits in DEVIOS. Go to UCLS2.

UCLS2 — If an output close is not desired, or if already closed, go to UCLS3. If DECtape save mode (2) was used to write the file, go to UCLS3. If not dump mode, go to UCLSBO to close buffered output; otherwise, dispatch to the device service routine output close function (dispatch index "DCL"), then return to UCLS3.

UCLSBO — If an output buffer was not set up or never referenced, go to UCLS3; otherwise, fall into UCLS2A.

UCLS2A — If DEVOAD addresses an empty buffer, go to UCLS2B; otherwise, clear the device error bits and call OUT (the OUTPUT UUO routine) to output this buffer. WAIT1 is called to stall until the buffer is emptied and the device has advanced the buffers. If no device error occurred, return to UCLS2A (this loop will continue until all full buffers have been output). If a device error is detected, go to UCLS2B.

UCLS2B — Dispatch to the device service output close routine (dispatch index "DCL"). Then clear the ring use bit and item count in the ring header. WAIT1 is called to be sure that the device is inactive, then UCLS3 is entered.

UCLS3 — Stores DEVDAT (in which the UUO bits have been modified) back into USRJDA for this channel, then returns to the calling routine.

See Figure 7 for a flow chart of the CLOSE operator.

## RELEASE OPERATOR

RELEA0, RELEA1, RELEA2, RELEA3 — The routine CLOSE1 is called to close both the input and output sides of the channel. WAIT1 is then called to be sure activity has ceased. Go to RELEA5.

RELEA5 — Dispatches to the device service release routine (dispatch index "DRL"). Then clears the active bit in DEVIOS and the USRJDA entry for this channel. Fetches the number of the highest channel in use from USRHCU.

RELEA4
RELE4A — These sections of code perform two important functions while scanning USRJDA from the old highest channel down to 0. First, USRHCU is changed, if necessary, to point to the highest nonzero entry in the JDA table. Second, if it is discovered during the scan that the device just released in RELEA5 is also assigned on another channel, then no further release housekeeping is performed, and an immediate return occurs. (It is possible to INIT the input and output sides of a bidirectional device on two separate channels.) Otherwise, the DEVIAD and DEVOAD words in the DDB are cleared, and control goes to RELEA9.

RELEA9 — If the device is a disk or is not the system tape, go to RELEA7. If it is the system tape but has already been returned to the system, go to RELEA7; otherwise, clear out the system tape user word, decrement the request count, and set the available flag if someone is waiting. Go to RELEA7.

RELEA7 — Supplies the ASSPRG bit to RELEA6 (RELEA6 may also be called by the DEASSIGN command, in which case an ASSCON bit would be supplied).

RELEA6 — Clears the assignment bit supplied by the calling routine. If the device is still assigned by another means, an immediate return is made. If the device is now wholly deassigned (ASSCON and ASSPRG = 0), then the job number field of DEVCHR is cleared. If the device is DSK, the routine CLRDDB is called to return to free storage the space occupied by the DDB. Return to the calling routine.

See Figure 8 for a flow chart of the RELEASE operator.

## LOOKUP AND ENTER OPERATORS

These operators are extremely device dependent, most of the work being performed by the device service routine. Before dispatching to the device service routine, however, LOOKUP performs an input close, and ENTER performs an output close, with appropriate alteration of the device status bits.

See Figure 9 for a flow chart of the LOOKUP and ENTER operators.

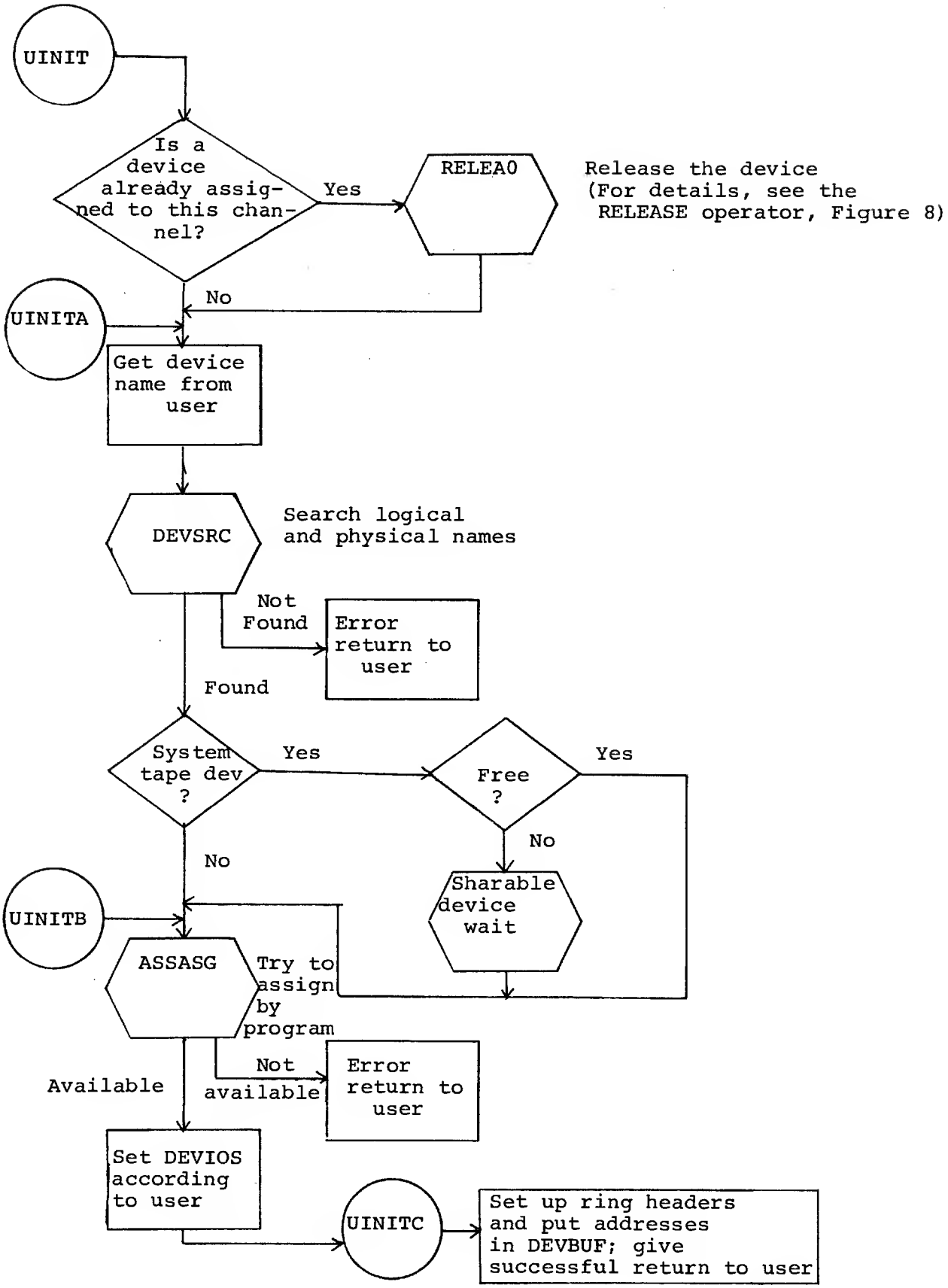



Figure 3. Flow Chart of INIT

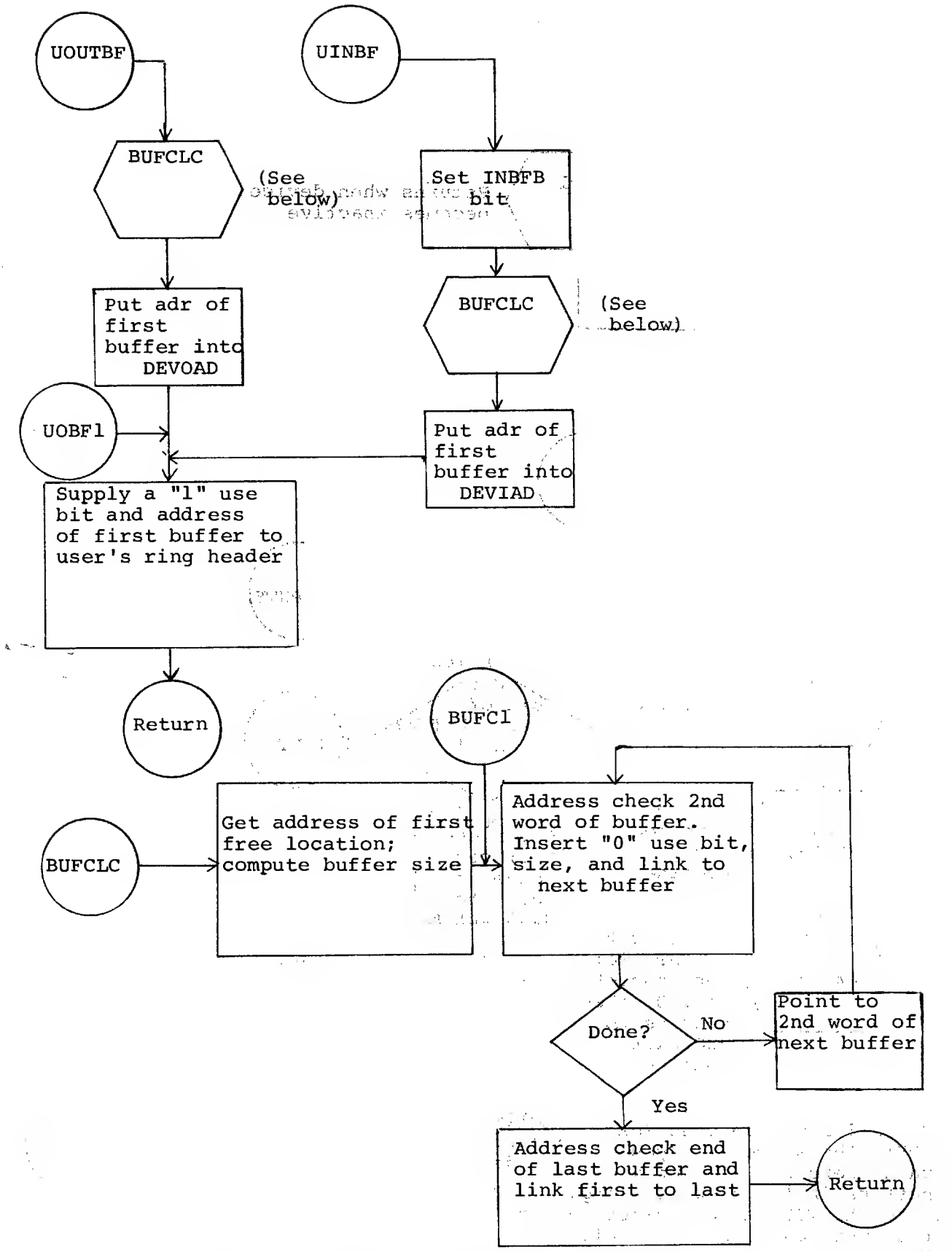



Figure 4. Flow Chart of INBUF, OUTBUF

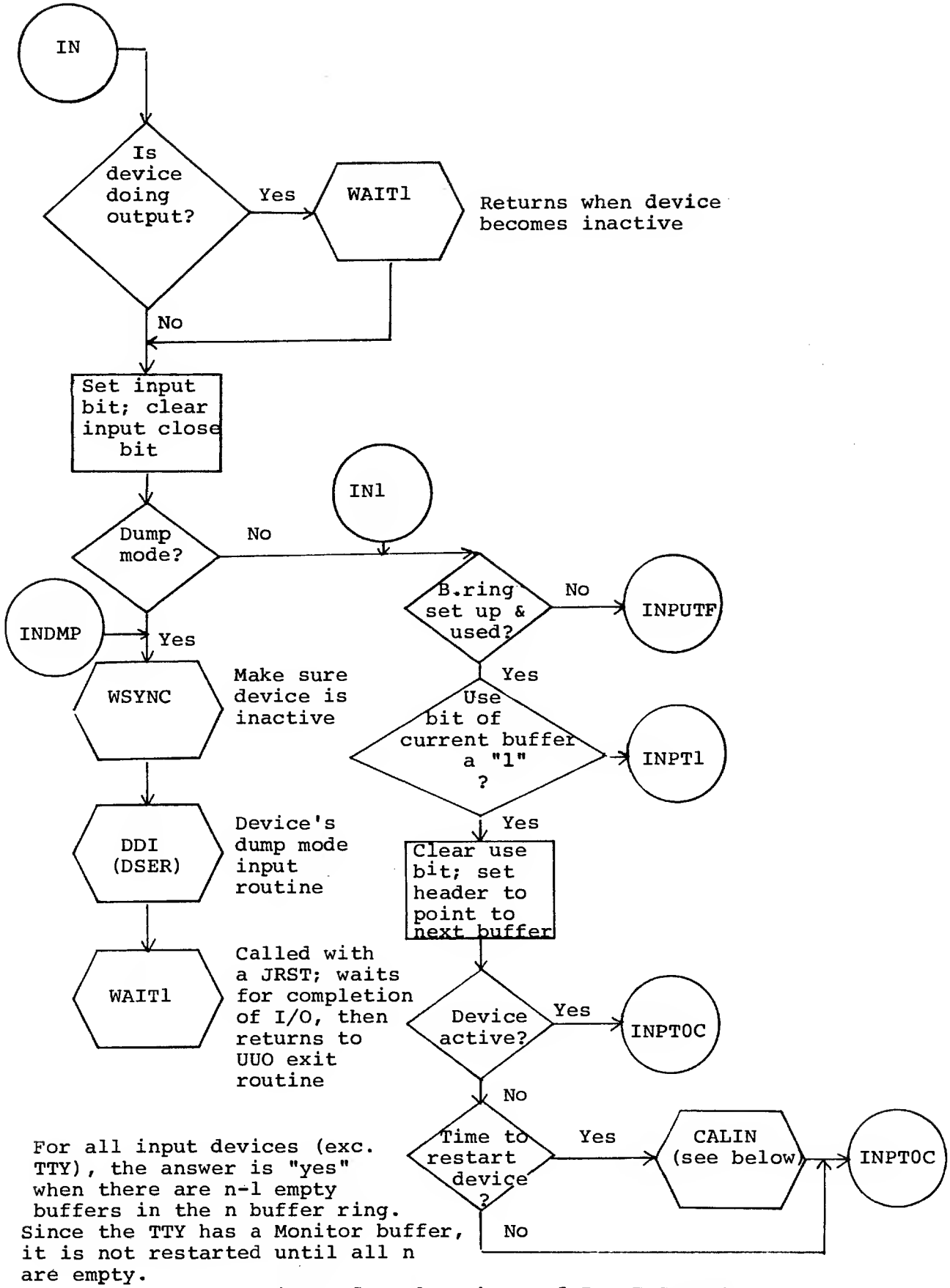



Figure 5. Flow Chart of INPUT Operator

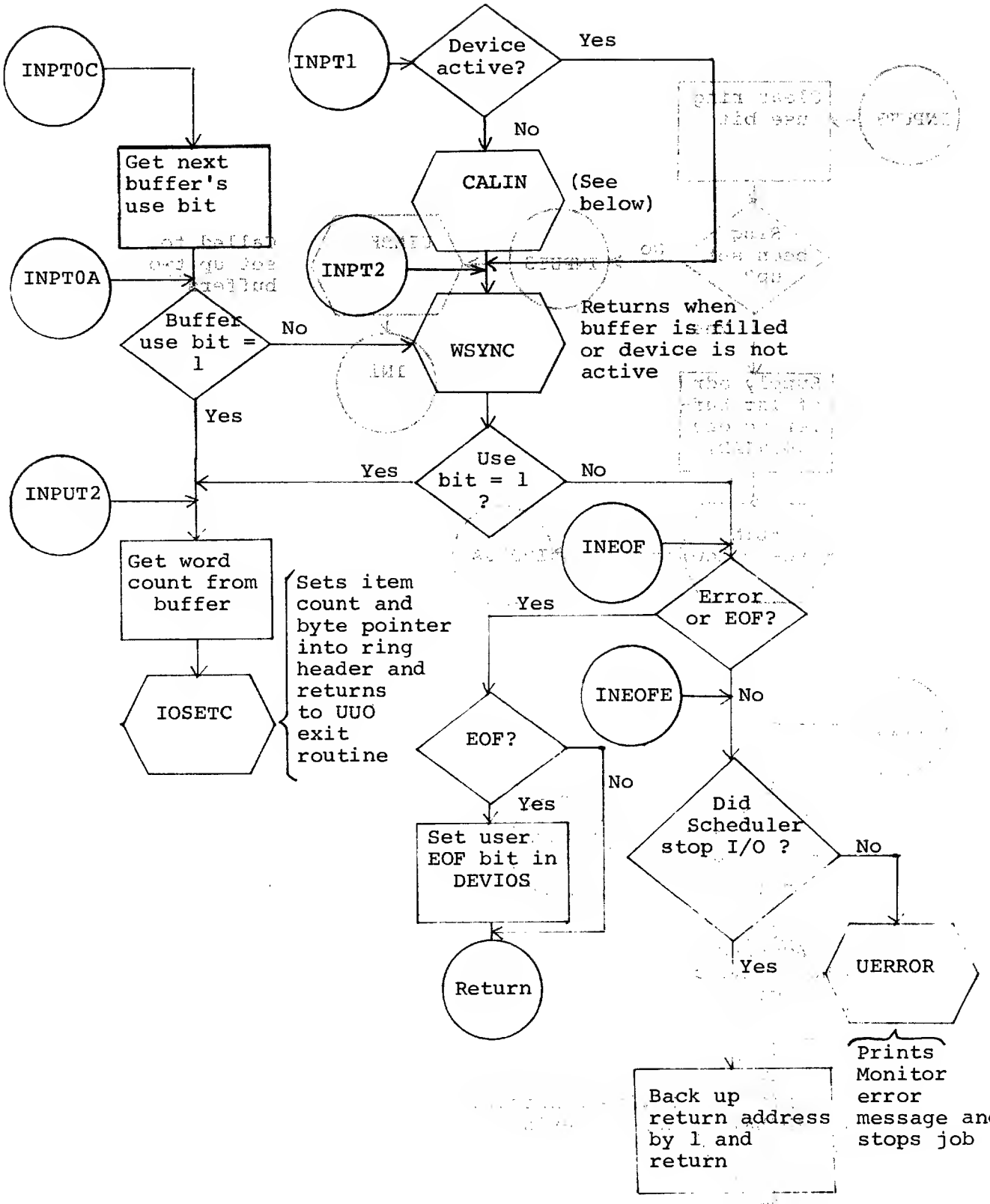



Figure 5 (Cont.) Flow Chart of INPUT Operator

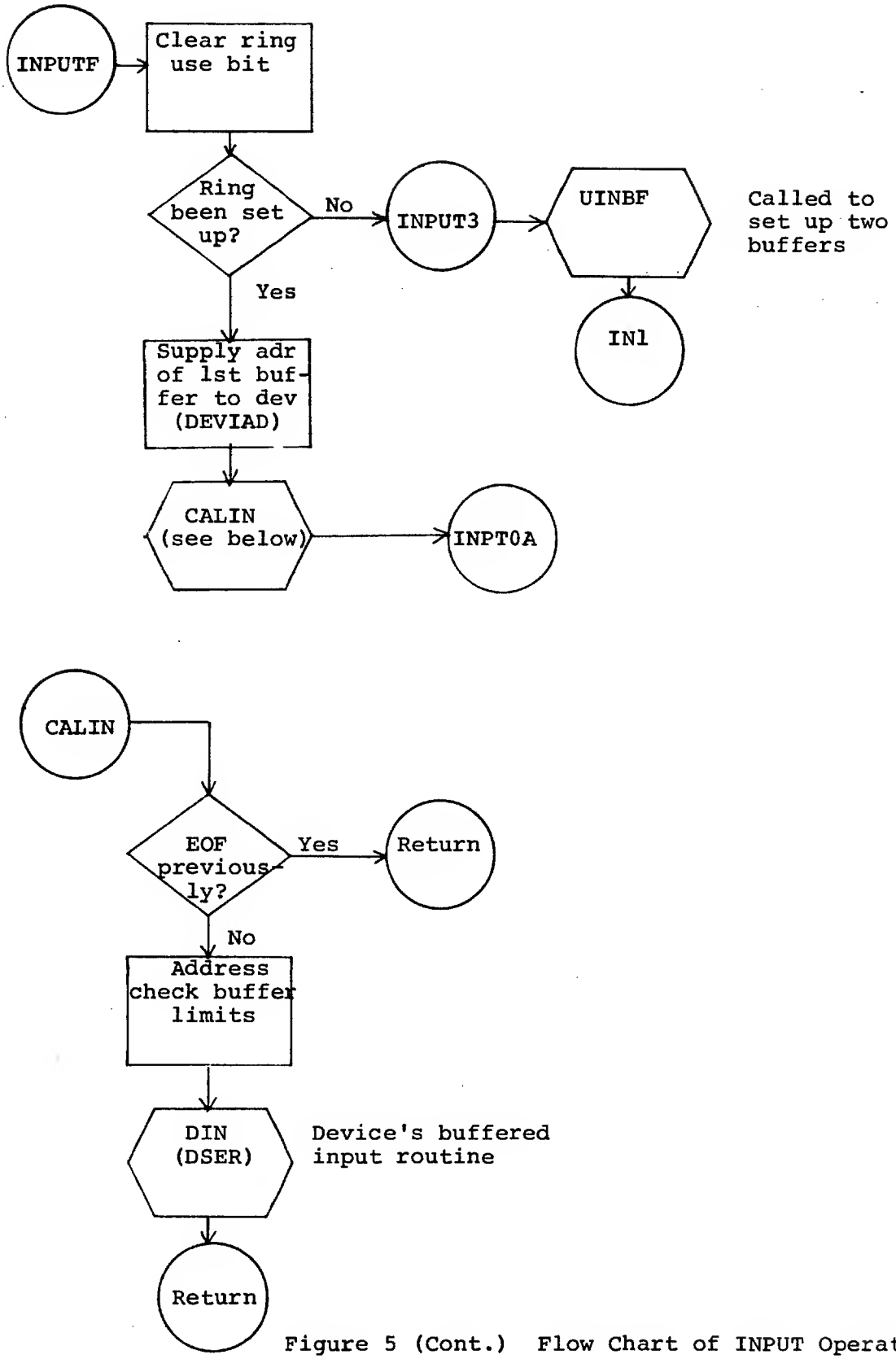



Figure 5 (Cont.) Flow Chart of INPUT Operator

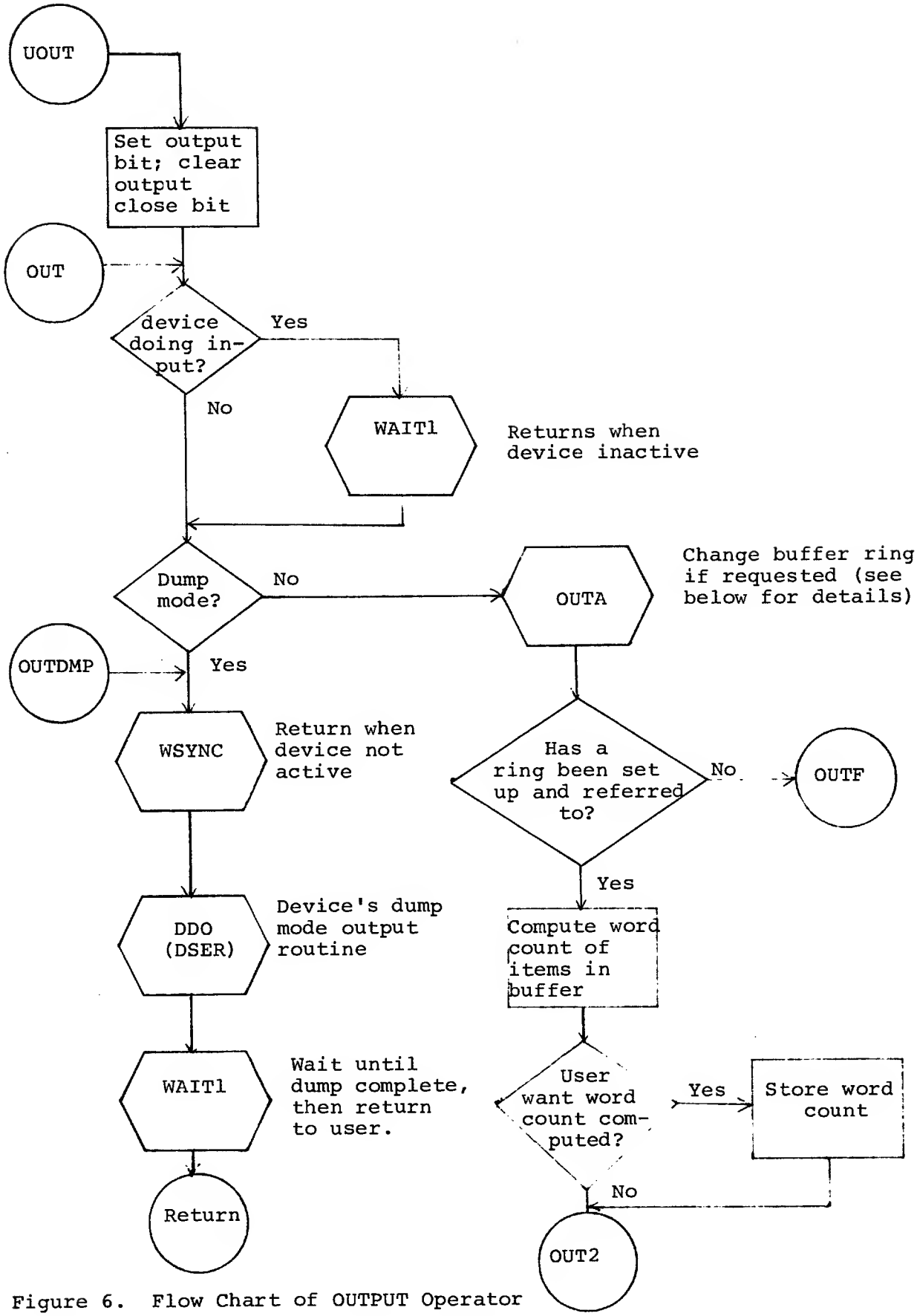



Figure 6. Flow Chart of OUTPUT Operator

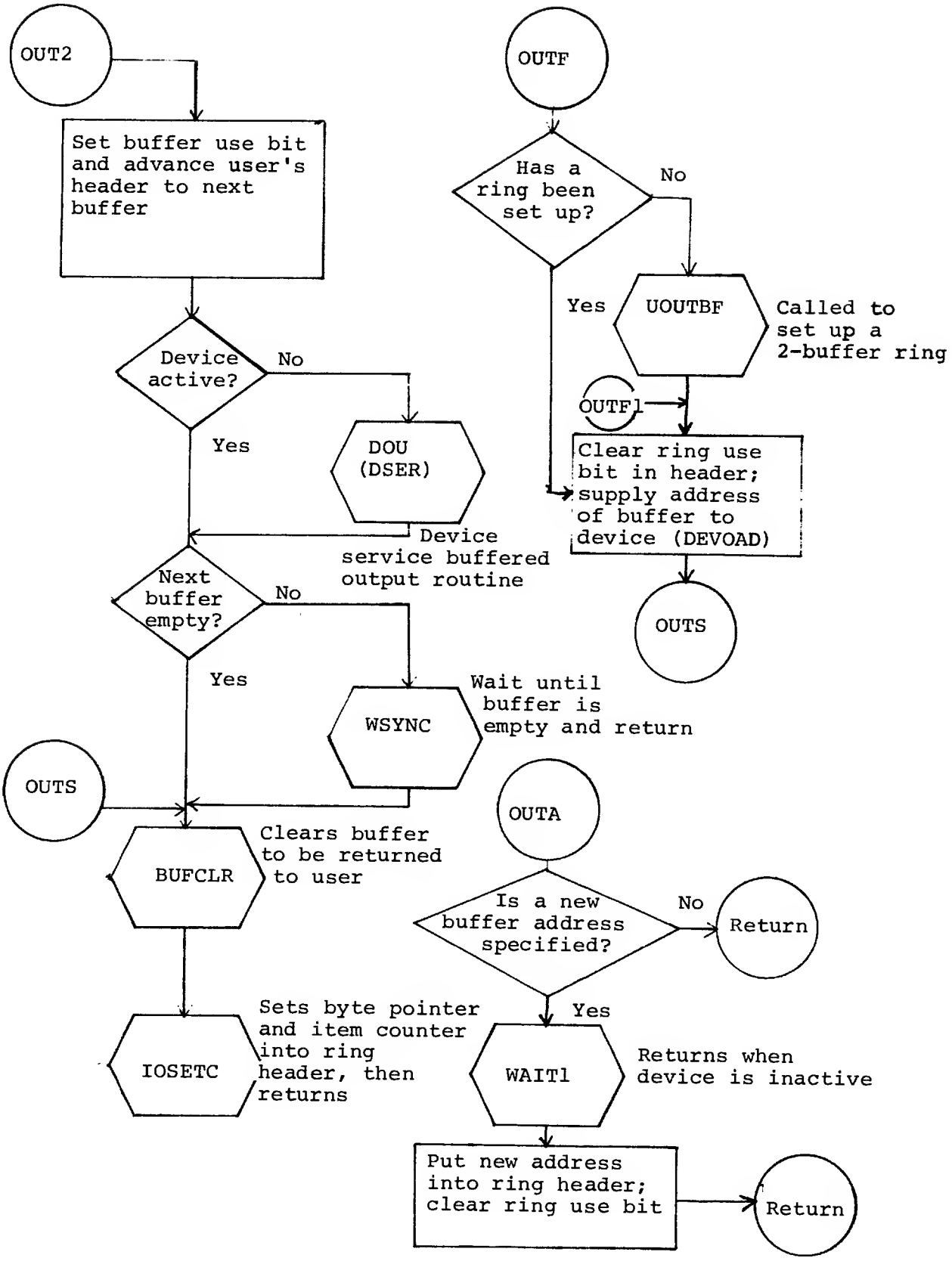



Figure 6 (Cont.) Flow Chart of OUTPUT Operator

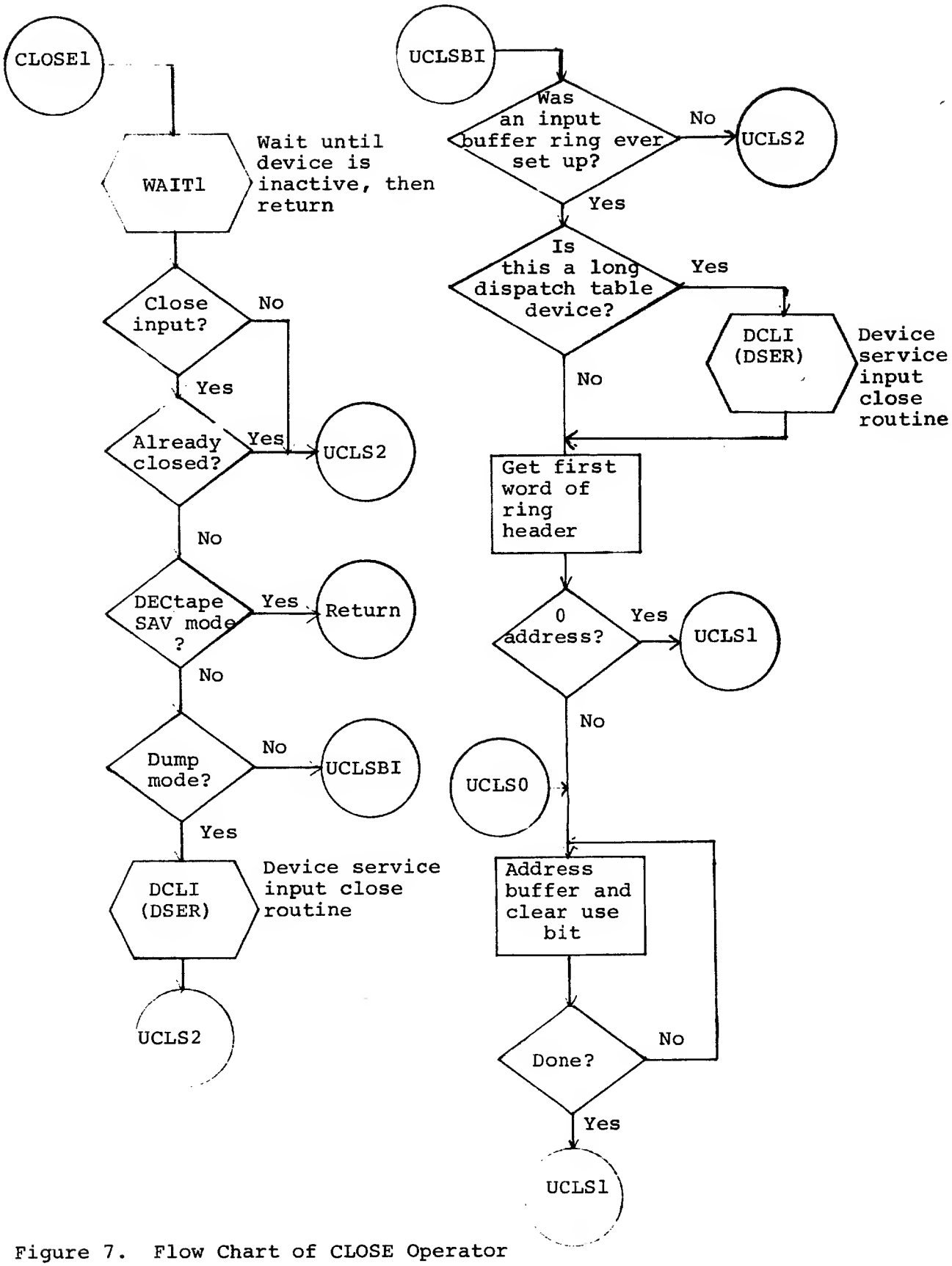



Figure 7. Flow Chart of CLOSE Operator

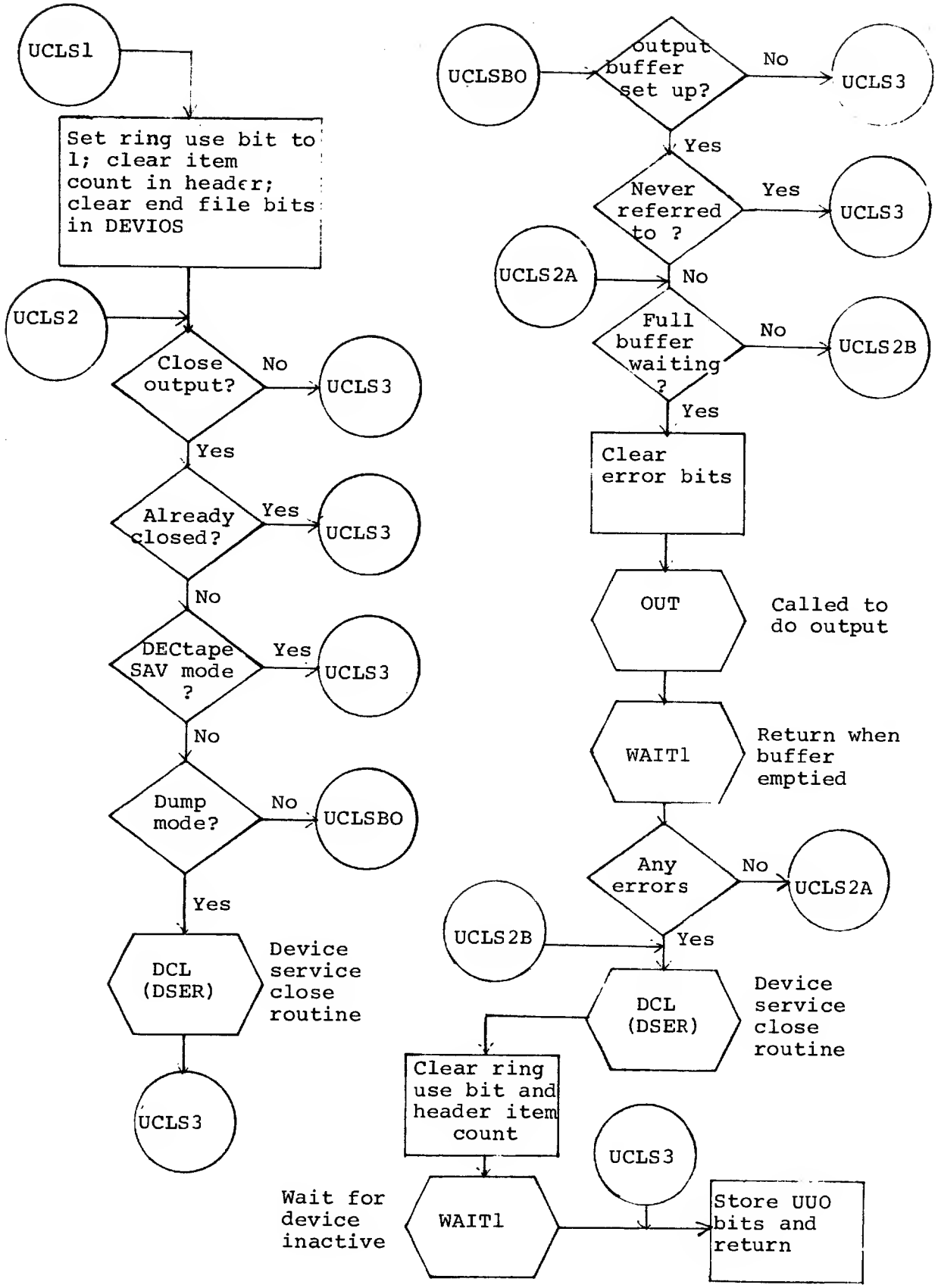



Figure 7 (Cont.) Flow Chart of CLOSE Operator

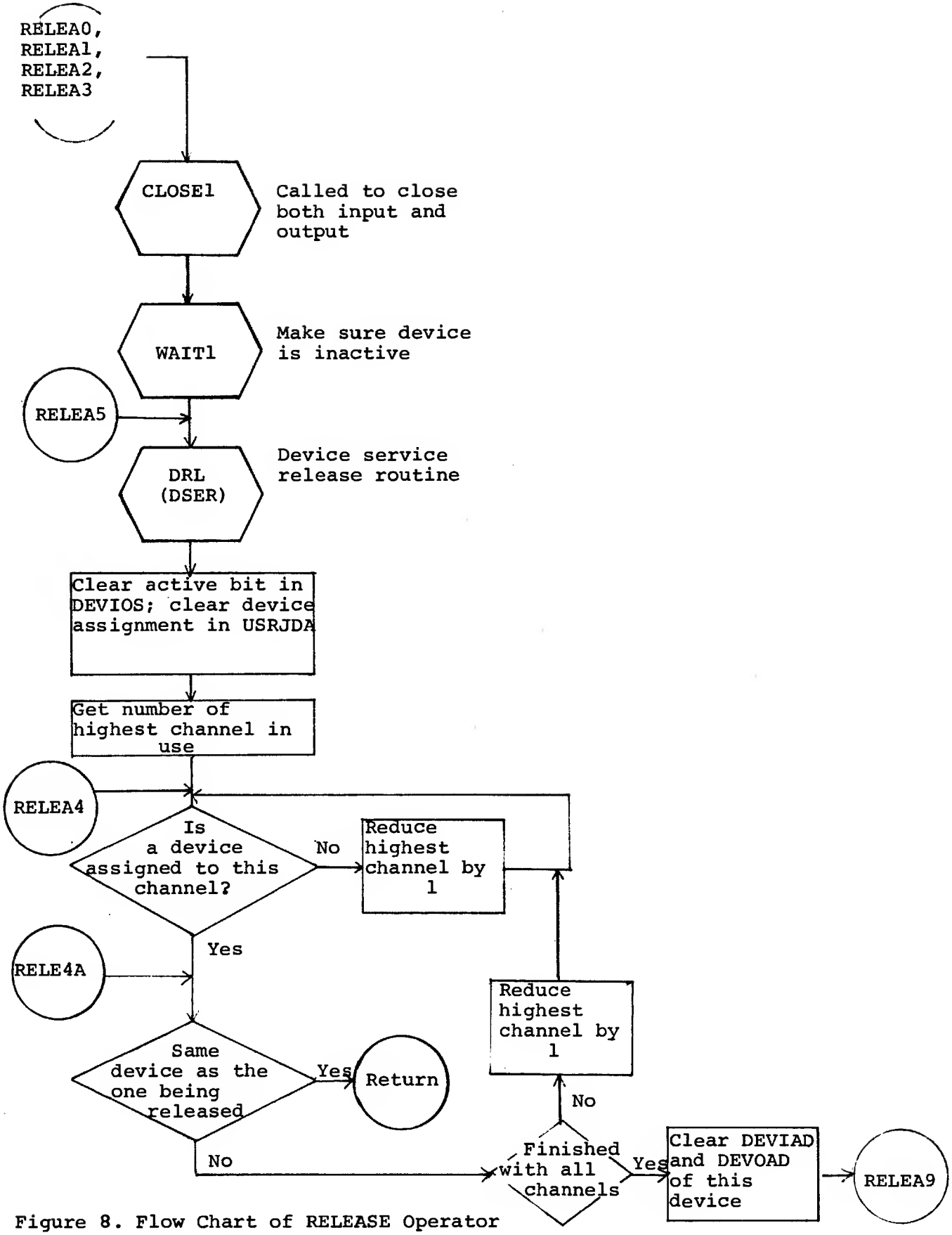



Figure 8. Flow Chart of RELEASE Operator

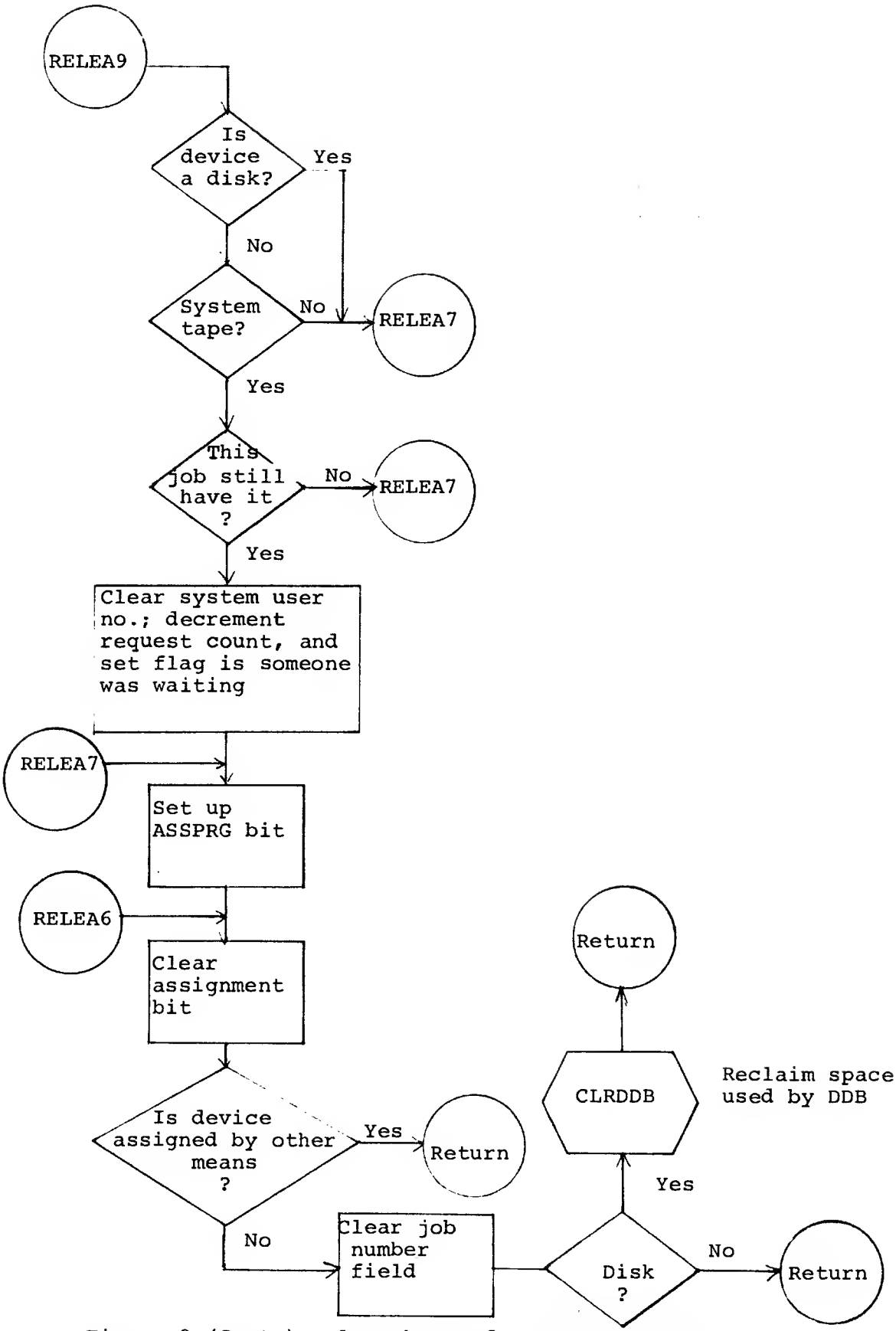



Figure 8 (Cont.) Flow Chart of RELEASE Operator

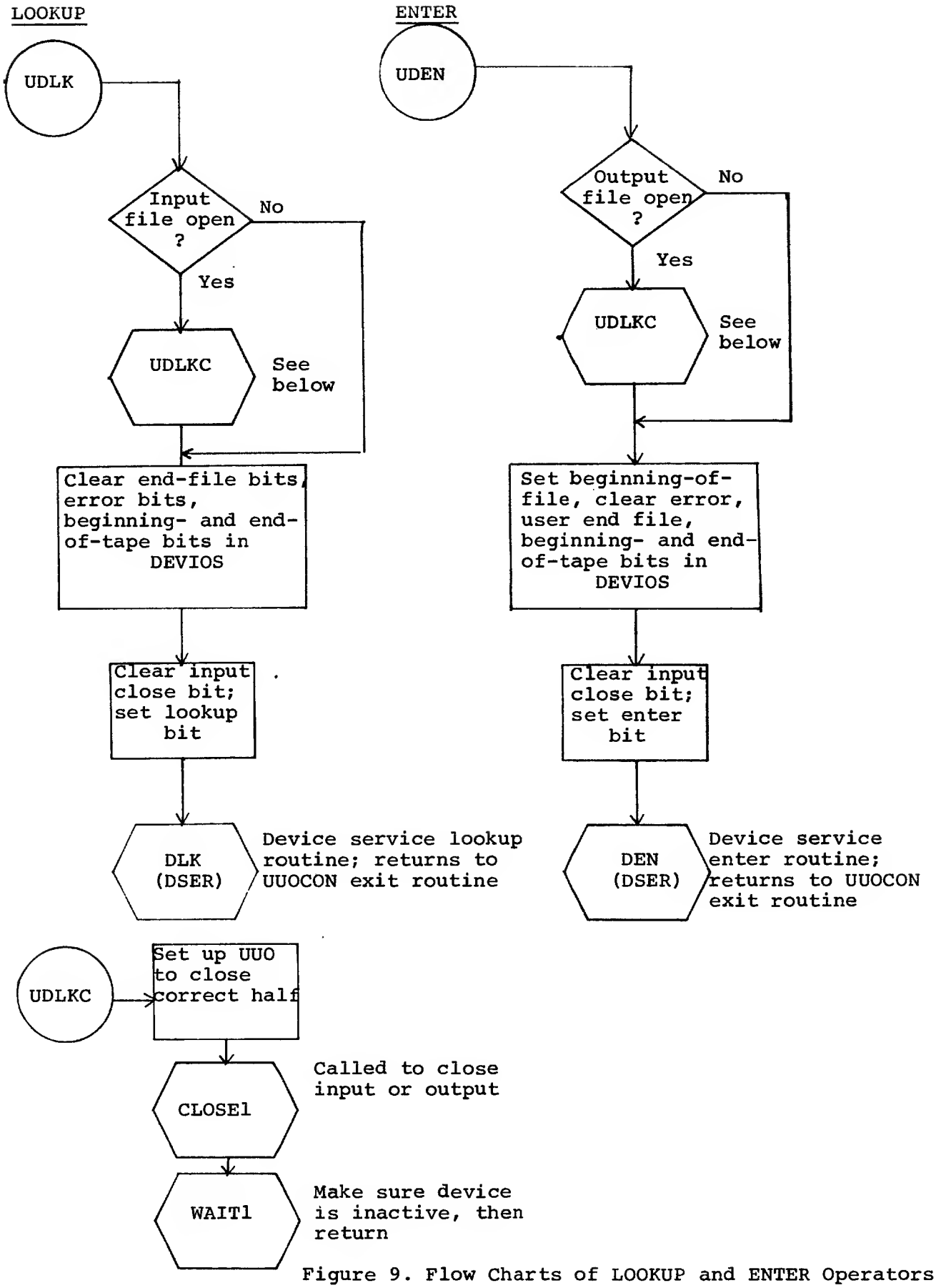



Figure 9. Flow Charts of LOOKUP and ENTER Operators

# III. DEVICE SERVICE ROUTINES

## DEVICE DATA BLOCKS

(See Figure 10)

The device data block (DDB) structure is the key to I/O handling on the UUO level in the PDP-10 Monitors. Each physical device is represented by a block of words beginning at devDDB, where dev is the 3-letter device mnemonic. The contents of the device data block completely describe a particular device at any given time; this description includes the physical characteristics of the device, the I/O status of the device, and the information required to link sections of the Monitor that communicate with one another while referring to the device described by the data block. While any routine is referring to a DDB, its address (devDDB) is kept in the accumulator DEVDAT, which is then used as an index register.

Each location in a DDB is known by a logical 6-letter mnemonic, which is defined in the System Parameter Tape to be a constant equal to the address of the location relative to devDDB (the address of the specific device data block). Thus, DEVxxx(DEVDAT) is the address of a specific word in a particular DDB, where DEVxxx represents the relative DDB location. Following is a description of the function of each location within the DDB, starting with the first word, DEVNAM (DEVNAM=0; others in ascending order.

DEVNAM — Contains the physical device name, left justified, in 6-bit ASCII (in the case of multiple devices, this causes the device number to fall left justified in the right half).

DEVCHR — Contains information giving the device assignment, hung device count, buffer size, binary device number (the bits set in each word are defined in the System Parameter Tape).

DEVIOS — Contains bits describing the current I/O status of the device. The left half is used only by the Monitor, while the right half becomes a user's device status register, which can be referred to by GETSTS, SETSTS, STATO, and STATZ UUO's (see Table 1 for DDB bit definitions).

DEVSER — Contains system linking information. The left half contains the address of the next DDB in a "chain" of all DDB's; the address of the first DDB in the chain is in the left half of DEVLST (a location in UUOCON), while the last DDB in the chain has zeroes in the left half of DEVSER. The right half contains the address of the Device Service Dispatch Table, which is referred to by UUOCON.

DEVMOD — The left half contains bits which, for the most part, describe the physical characteristics of the device; most of these are assembled as part of the DDB. These bits can be called by the user with a GETCHR or DEVCHR UUO (this is not to be confused with the DEVCHR DDB word - see above). The right half has bits indicating whether assignment of the device was by console command and/or by the INIT UUO, as well as bits reflecting which data modes are legal for the device (see Table 1).

DEVLOG — Contains the logical device name (left justified, in 6-bit ASCII) assigned by the user from the console Teletype. When executing the INIT UUO, which links the word in location USRJDA (UCHN) with a DDB, the Monitor scans the contents of DEVLOG through the DDB chain before trying to match the user's specified device with the contents of DEVNAM.

DEVBUF — Contains addresses of buffer headers associated with the device by INIT UUO; the left half contains the output header address, while the right half contains the input header address.

DEVIAD (or DEVADR) — Contains the address of the user's input buffer which is currently being filled (DEVIAD=DEVADR=7).

DEVOAD (or DEVPTR) — Contains the address of the user's output buffer currently being emptied.

NOTE: In the time-sharing Monitor, the accumulator used for relocation (PROG) is designated in the index register field of both DEVIAD and DEVOAD.

DEVCTR (or DEVFIL) — Contains item count for the buffer (same as the third word of user's buffer header). For directory devices which have long dispatch tables, this location is called DEVFIL and contains the 6-bit ASCII name of the file being referred to, while the next location (DEVEXT) contains the extension, if any, of the file.

There are devices that reserve more locations for the DDB's than those mentioned above, but these additional locations are required by the special characteristics of the particular device rather than by the system itself.

When a device service routine services a class of multiple devices (e.g., DTASER services DTA0, DTA1, ....etc.), only the DDB of the first device, DEV0, is assembled into the routine. The rest of the blocks are loaded outside the routine by Build, being modeled after the DTA0 DDB and being linked in the chain via DEVSER. Build determines the number of DDB's to create for a device service routine from responses received during the console dialogue.

System Index

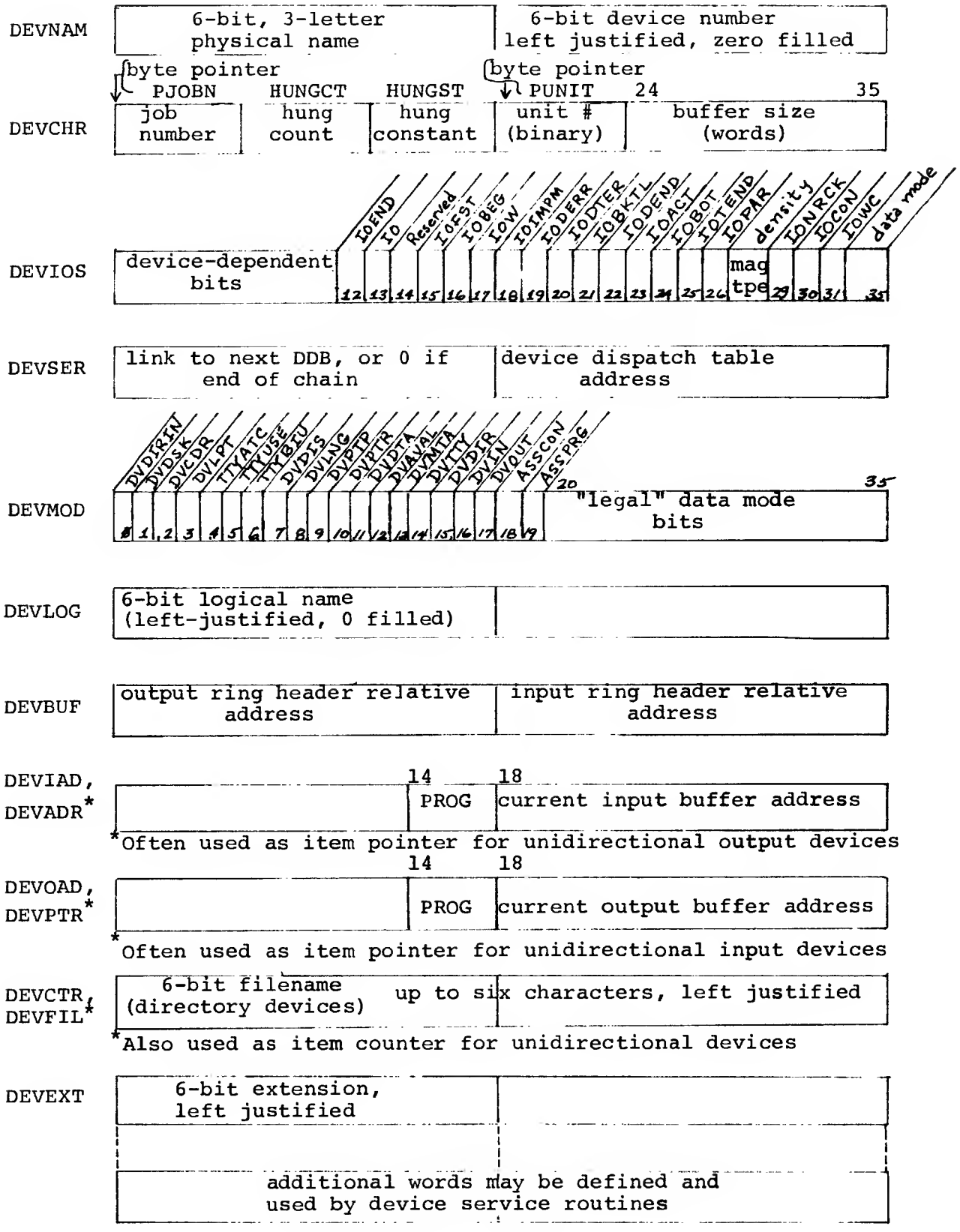



Figure 10. Device Data Block (DDB)

Table 1. Device Data Block (DDB) Bit Definitions

| | | |
|---|---|---|
| DEVIOS | | I/O Status |
| | IOEND | Set at interrupt level by input device when end of file recognized |
| | IO | Direction of transfer: Out = 1, In = 0 |
| | IOFST | Set by service routine to indicate that next interrupt is first item of a buffer. |
| | IOBEG | Set by INIT or ENTER operator to indicate a "new file" |
| | IOW | Set when a job is placed in an I/O Wait State |
| | IOIMPM | Improper mode detected by input service routine |
| | IODERR | General device error bit |
| | IODTER | Device data error bit |
| | IOBKTL | "Data block too large" error |
| | IODEND | End of file (to user) |
| | IOACT | Device active, expecting interrupts |
| | IOBOT | Beginning of magnetic tape |
| | IOTEND | End of magnetic tape |
| | IOPAR | Write even parity (mag tape command) |
| | IONRCK | Read with no reread |
| | IOCON | Discontinuous I/O if set to 1. Device stops after filling or emptying each buffer |
| | IOWC | Inhibit system computation of word count for output device |
| DEVMOD | | Device characteristics and legal data modes |
| | DVDIRIN | Directory is in core |
| | DVDSK | Device is a disk |
| | DVCDR | Device is a card reader |
| | DVLPT | Device is a line printer |
| | TTYATC | This Teletype is attached to a job |
| | TTYUSE | This Teletype is in user mode |
| | TTYBIU | Teletype DDB in use |

Table 1 (Cont.) Device Data Block (DDB) Bit Definitions

| | |
|---|---|
| DVDIS | Device is a display |
| DVLNG | Device service routine has a long dispatch table |
| DVPTP | Deviee is a paper tape punch |
| DVPTR | Device is a paper tape reader |
| DVDTA | Device is a DECtape |
| DVAVAL | Device is available (set by DEVCHR UUO) |
| DVMTA | Device is a magnetic tape |
| DVTTY | Device is a Teletype |
| DVDIR | Device has a file directory |
| DVIN | Device is capable of doing input |
| DVOUT | Device is capable of doing output |
| ASSCON | Device has been assigned by a console command |
| ASSPRG | Device has been assigned by a program (INIT or OPEN) |

Bit (35 - n) is a 1 if mode n is legal for this device

| Mode | n (decimal) |
|---|---|
| ASCII | 0 |
| ASCII line | 1 |
| DECtape SAV | 2 |
| Image | 8 |
| Image Binary | 11 |
| Binary | 12 |
| Image Dump | 13 |
| Dump Records | 14 |
| Dump | 15 |

## Dispatch Table

The Monitor dispatches to device-dependent coding via a dispatch table located in that coding. The base address of this table exists in accumulator DSER during the processing of an I/O operator. The dispatch is usually performed by a PUSHJ PDP, INDEX (DSER), where INDEX is a constant used to select the appropriate entry of the table. See Figure 4 for an illustration of such indices. A "basic" dispatch table has six entries and is sufficient for service routines of "simple" physical devices such as card readers, tape punches, and line printers. Devices which require directory manuevers or complex activities in file positioning use a so-called "long dispatch table" containing 17 entries (including the "basic" ones). Examples of these are DECtape, magnetic tape, and disk. Before attempting to dispatch on a "long-type" UUO, Monitor checks the DVLNG bit in the DEVMOD word (see routine DISP1 in UUOCON, for example).

Table 2. Device Service Dispatch Table Entries

| | System Index | | Purpose |
|---|---|---|---|
| "BASIC" | -2 | DINI | Device and service routine initialization |
| | -1 | DHNG | "Hung device" action |
| | 0 | DRL DEVDSP: | Release (table base address) |
| | 1 | DCL, DCLO | Close, close output |
| | 2 | DOU | OUTPUT Operator |
| | 3 | DIN | INPUT Operator |
| "LONG" | 4 | DEN | ENTER Operator |
| | 5 | DLK | LOOKUP Operator |
| | 6 | DDO | Dump Mode output |
| | 7 | DDI | Dump Mode input |
| | 10 | DSO | USETO Operator |
| | 11 | DSI | USETI Operator |
| | 12 | DGF | UGETF Operator |
| | 13 | DRN | RENAME Operator |
| | 14 | DCLI | Close input (dump mode) |
| | 15 | DCLR | CALL X,[SIXBIT/UTPCLR/] |
| | 16 | DMT | MTAPE Operator |

## Basic Operations

This section attempts to describe, in summary fashion, the actions performed by the device service routine upon receiving one of the six "basic" dispatches.

1. Initialization (Index DINI)

Entered from SYSINI Monitor initialization when Monitor is first loaded or upon certain restarts. The service routine should set the hardware control unit to some known free state (usually a CONO DEV, 0). The

routine may also have to preset its own software "flag" registers (the mask bits for interrupt level "CONSOing" are usually kept in a register, DEVCON, which should now be cleared). Return to the calling routine is via a POPJ PDP,

2. Hung Timeout (Index DHNG)

Entered from routine DEVCHK at clock interrupt level (refer to CIP5 in CLOCK). When a device is started by an INPUT or OUTPUT operator and each time an interrupt is serviced for this device, the HUNGCT field of DEVCHR is set to the value HUNGST. Every second, the HUNGCT field of all active devices is examined and, if nonzero, decremented. If decrementation causes HUNGCT to become zero, this dispatch is made (preloading of HUNGST with zero will prevent this from ever occurring).

Some device routines use this entry to perform a release or initialization, but tnere seems to be no clear-cut rule as to whether something should be done. Upon return (via a POPJ), the Monitor types out an informative message on the user's console and places the job in an error stop state.

3. RELEASE Operator (Index DRL)

Entered from RELEA5 in UUOCON. If the service routine controls a single unit device (paper tape reader, card punch, etc.), the hardware is released by an action similar to that described in (1) above. If it is capable of controlling multiple units (e.g., magtape), the control unit should not be disturbed as it is likely servicing another job's I/O. The service routine for a directory device (DECtape, disk) should use this entry to write out a fresh copy of the directory if it has changed since it was first read into core. Thus, the releasing action may range from an immediate return (POPJ) to an actual output data transfer with consequent placing of the job in I/O Wait, then returning.

4. CLOSE Operator (Index DCL, DCLO)

Entered from UCLS2 or UCLS2B in UUOCON for closing either dump or buffered output. "Basic" devices are never entered when an input close is performed; this occurs only for "long dispatch table" devices at index DCLI.

For buffered output modes, an attempt should be made to output a possible partially filled buffer with a PUSHJ PDP,OUT (this does no harm if there is no more output to be done. Also there is no possibility at this point of there being more than one buffer to flush because the device independent part of CLOSE has taken care of all full buffers). WSYNC may now be called to allow completion of activity if the service routine wants to perform some additional operations in closing the file. If not, a POPJ will return to the CLOSE coding in UUOCON, which does a wait before returning to the user.

Additional operations include end-of-file marking and formatting. Examples: Magnetic tape service writes two end-file marks and backspaces over one of them. Line printer service sends out a carriage return, form feed combination. Paper tape punch service punches about 13 in. of blank tape.

5. OUTPUT Operator (Index DOU)

Entered from OUT2 in UUOCON to start device doing buffered output. The activities of file positioning, formatting, and data transfer all take place at interrupt level. The job of the OUTPUT routine is to condition the interrupt level coding (by setting software switches, counters, etc.) to perform the desired activity and then to prime the hardware control unit so that an interrupt will occur. The OUTPUT routine must also set some indicators so that other sections of the Monitor will know that this device has been made active for OUTPUT.

If desired, the first dispatch (beginning of file) to the output routine may be detected by testing the bit IOBEG in accumulator IOS. This bit is set by an INIT operator and should be cleared by the service routine. For example, detection of this bit causes paper tape punch service to output a fanfold of blank tape before the data of a file. The first output call is also used to get the address of the first buffer from word 1 of the user's ring header and store it in DEVOAD of the device data block. In IOS, the IO bit should be set to 1 (output) and the IOFST bit set to 1 (first item of a buffer).

As part of initialization, the byte pointer used to get data from a user buffer is set up. IOS contains the data mode supplied as part of initial status. When called by PUSHJ PDP, SETBYT, this routine will return in TAC a partial byte pointer containing a size field according to the data mode and "PROG" in the index field. The left half of TAC may now be stored in the pointer location of the device data block. The right half is usually filled in at interrupt level each time a new buffer is b egun (detection of IOFST = 1).

When all IOS bits have been set up, the routine SETACT may be called with a PUSHJ. This coding sets the active bit, IOACT, stores IOS into the device data block and initializes the hung count (HUNGST→HUNGCT) before returning.

The next operation is to start the physical device with a CONO. Simple output devices are started by supplying an interrupt channel address and setting the "DONE" (ready for data transfer) flag. It may also be necessary to supply other conditions to the hardware, but the former are essentials. The CONSO instruction issued at interrupt level to test for expected flags may pick up a mask indirectly to allow the same instruction to test different conditions at different times. If desired (and this is typical), the mask bits should be placed in this location at the time the device is started. A macro STARTDV defined in the file "S" may be used as follows.

Place the desired CONO bits in the right half of TAC and the CONSO mask bits in the left half. Then write STARTDV XYZ, where XYZ is the device mnemonic (first three letters of service routine title, XYZSER). This macro expands as follows.

```
STARTDV  XYZ↑ EXTERNAL PIOFF, PION
CONO PI, PIOFF
CONO XYZ, (TAC)
HRLM TAC, XYZCON
CONO PI, PION
```

Location XYZCON must, of course, be defined within the service routine.

Having started the device, return to UUOCON with a POPJ PDP,.

6. INPUT Operator (Index DIN)

Entered from CALIN in UUOCON to start the device doing buffered input. While the actual data transfers will take place at interrupt level, the job of the INPUT coding is to condition the interrupt coding to perform the desired actions and then to start the device so that an interrupt will occur. The first input call for a file (IOBEG = 1) is used to get the address of the first buffer in the ring from word 1 of the user's ring header and store it in DEVIAD. The desired bits of DEVIOS are manipulated in IOS, then stored with a PUSHJ PDP, SETACT which also turns on the IOACT bit and resets the hung timeout count. The device may then be started using the STARTDV macro as described under (5). When starting an input device, the CONO bits assign a PI channel number and turn on the BUSY flag. The latter sets the physical device in motion to gather the first word or character from the input medium, at completion of which the DONE flag sets, causing the interrupt.

## INTERRUPT-LEVEL OPERATIONS

### Interrupt Channel Routines CHAN and NULL

The Monitor contains one of these routines for each of the seven priority interrupt levels. A CHAN routine exists for a given level if there is at least one service routine assigned to that level (by Build). A NULL routine exists for each unused level. At initialization time, the routine LINKSR in ONCE ONLY CODE places the instruction JSR CHn in each location 40+2n (42, 44, . . . 56). The NULL routine defines CHn as a location to contain the PC word and the next instruction attempts to dismiss this spurious interrupt with a JEN @CHn.

A CHAN routine contains a like entry point, but the next location contains a JRST to the interrupt entry of the first service routine built on this PI level. A CHAN routine also contains a subroutine, SAVCHn (called by a JSR) to save accumulators 0 through 10 and set up a pushdown pointer, and a subroutine XITCHn to restore accumulators and dismiss the interrupt. The pushdown list and accumulator storage locations are in the body of the CHAN routine.

When a service routine is coded, it is not known what PI level will be

assigned at Build time; therefore, there is a standard symbology used to refer to the CHAN entries. If XYZ is the device mnemonic, the following symbols (declared EXTERNAL in the device service routine) will be equated by Build.

```
XYZCHN = PI channel number, 1 through 7
XYZCHL = CHn, interrupt PC word
XYZSAV = SAVCHn, AC storage subroutine
XYZXIT = XITCHn, AC restore and dismissal
```

There are three ways to exit from an interrupt routine. If the routine has saved and restored all accumulators within its own coding, the dismissal may simply be JEN @XYZCHL. If the initial part of the routine called XYZSAV to save accumulators and set up a pushdown pointer, a JRST XYZXIT will cause restoration and dismissal. Alternately, an "extra" POPJ PDP, can be used because the pushdown list is assembled with the address XITCHn as its 0 entry.

## Interrupt Service

The interrupt level coding of a device service routine handles data transfers and error conditions. The routine is responsible for transmission of one byte between a user's buffer and the file, and for advancing buffers when necessary. The routine must stop the hardware device when no buffer is available (device has caught up with the user) or, conversely, take the job out of a Wait if the latter condition is detected upon completion of a buffer (user caught up with device). The flow charts in this section (Figures 11 and 12) describe the general logic used for interrupt level processing. In practice, some alterations in flow and wide variations in coding technique will occur because of differences in device speeds and hardware buffering. We suggest that the reader study the paper tape reader service (PTRSER) and paper tape punch service (PTPSER) routines, which reveal the coding techniques that support the functions outlined in these flow charts.

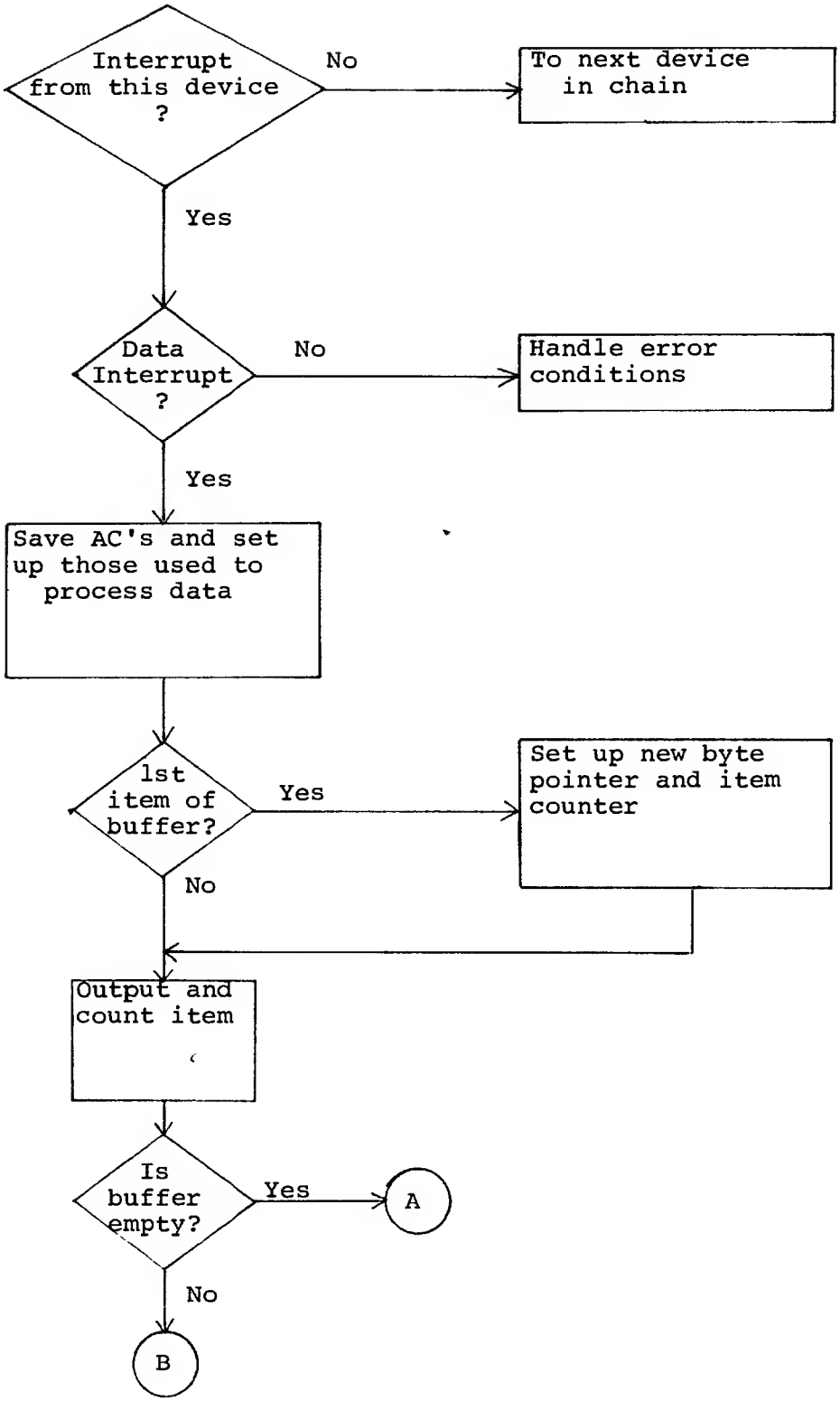



Figure 11. General Flow for Output Interrupt Routine

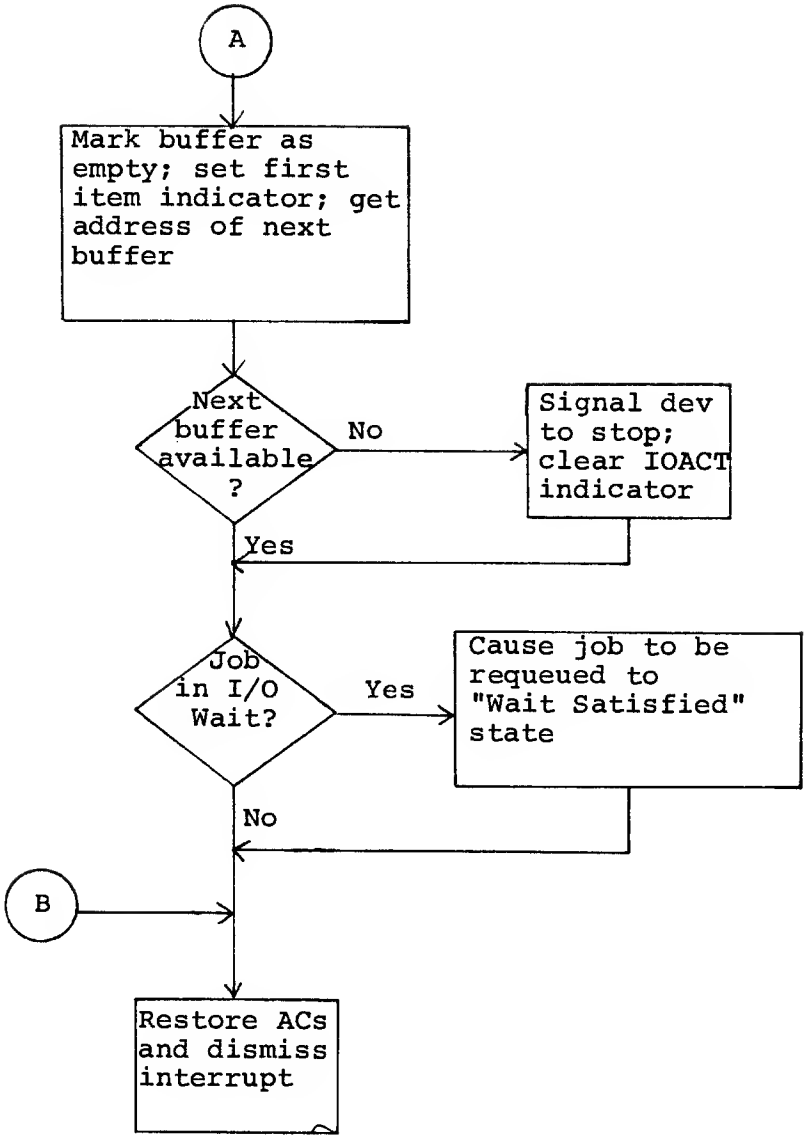



Figure 11 (Cont.) General Flow for Output Interrupt Routine

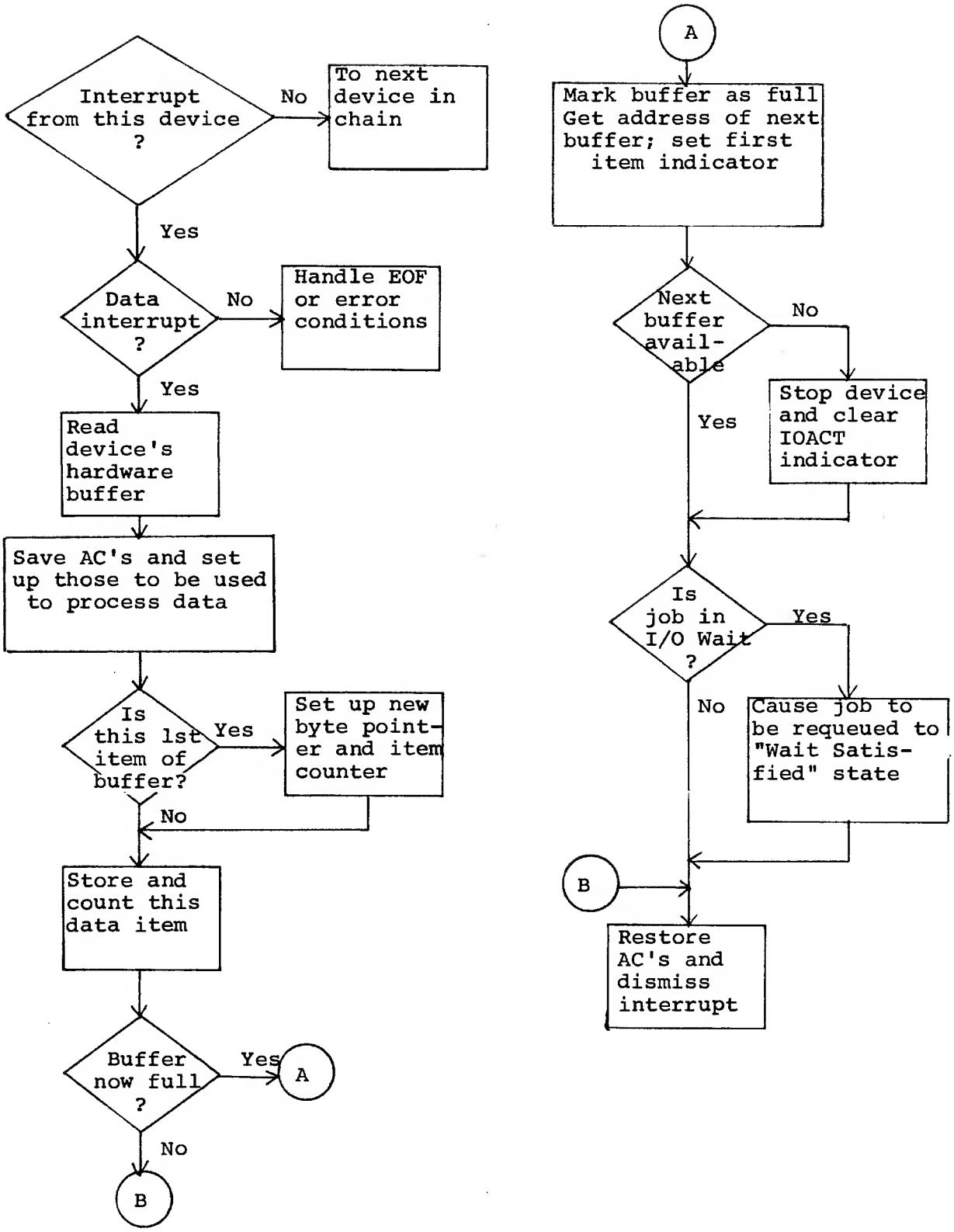



Figure 12. General Flow for Input Interrupt Routine

Table 3

MONITOR UUO'S

| Octal | Mnemonic | Description |
|---|---|---|
| 040 | CALL | Extended operation code (see Table 4) |
| 041 | INIT (D) | Allocate device with parameter in following words; error return at 3, normal at 4 |
| 042<br>043<br>044<br>045<br>046 | Reserved for installation use | |
| 047 | CALLI | Immediate mode extended operation code (see Table 4) |
| 050 | OPEN (D) | Allocate device; parameter block at E; skip if no error |
| 051<br>052<br>053<br>054 | Reserved for future DEC use | |
| 055 | RENAME (D) | Change file parameters to block at E; skip if no error |
| 056 | IN (D) | Input buffer; use buffer or command list at E ($\neq$0); skip if no error |
| 057 | OUT (D) | Output buffer; use buffer or command list at E ($\neq$0); skip if no error |
| 060 | SETSTS (D) | Wait for device inactive; load device status word with E |
| 061 | STATO (D) | Skip if any device status word bit masked by a 1 in E is a 1 |
| 062 | GETSTS (D) | Store device status word in E |
| 063 | STATZ (D) | Skip if all device status word bits masked by a 1 in E are 0 |
| 064 | INBUF (D) | Set up a ring of E standard size input buffers |
| 065 | OUTBUF (D) | Set up a ring of E standard size output buffers |
| 066 | INPUT (D) | Input buffer; use buffer or command list at E if $\neq$ 0 |

Table 3 (Cont.)

| Octal | Mnemonic | Description |
|---|---|---|
| 067 | OUTPUT (D) | Output buffer; use buffer or command string at E if ≠ 0 |
| 070 | CLOSE (D) | Finish I/O and close file; E = 0 - input and output, 1 - input, 2 - output, 3 - neither |
| 071 | RELEAS (D) | CLOSE input and output files and deallocate device |
| 072 | MTAPE (D) | Magnetic tape positioning (see below) |
| 073 | UGETF (D) | Store number of free DTA blocks in E |
| 074 | USETI (D) | Set DTA or DSK to input block E next |
| 075 | USETO (D) | Set DTA or DSK to output block E next |
| 076 | LOOKUP (D) | Select input file, parameter block at E; skip if no error |
| 077 | ENTER (D) | Select output file, parameter block at E; skip if no error |

NOTES:

1. I/O is performed by associating a device, a file, and a buffer ring or command list with one of a user's I/O channels (D).

2. MTAPE Commands:

| | | | |
|---|---|---|---|
| 1 | Rewind | 11 | Rewind and unload [1] |
| 2 | Write end of record | 13 | Write 3 in. of blank tape |
| 6 | Skip record | 16 | Skip file |
| 7 | Backspace record | 17 | Backspace file |
| 10 | Skip to logical end of tape | | |

[1] Documented but not implemented (hardware incompatible)

3. (D) - Channel number used (in AC field)

Table 4

CALL [SIXBIT/name/] and CALLI n

| Name | n | Description |
|---|---|---|
| RESET | 0 | Terminate user's I/O, user I/O mode; deallocated unASSIGNed dev |
| DDTIN (AC) | 1 | Wait for character, load buffer (address in AC) with characters typed since last DDTIN |
| | 2 | Not presently used |
| DDTOUT (AC) | 3 | Wait until output complete; type characters in buffer (address in AC) |
| DEVCHR (AC) | 4 | Load AC with device characteristics of device whose SIXBIT name is in AC |
| | 5<br>6<br>7 | Not presently used |
| WAIT (D) | 10 | Delay running until device inactive |
| CORE[1] (AC) | 11 | Change core assigned to number of blocks in AC (0 = no change); skip if granted. AC contains highest address |
| EXIT | 12 | RELEASe all I/O devices, type "EXIT, ↑C" on console; console enters Monitor mode |
| UTPCLR (D) | 13 | Clear (DTA) directory |
| DATE (AC) | 14 | Load 12-bit date in AC (right justified) |
| LOGIN[2] (AC) | 15 | Read n words from system file, pointer in AC (-n, TABLE) |
| APRENB[1] (AC) | 16 | Enable processor traps to user; AC contains enable bits in CONO APR, format |
| LOGOUT[1] | 17 | RELEASe all I/O devices, return job number, core, and devices to Monitor pool; do bookkeeping[3] |
| SWITCH (AC) | 20 | Load AC with processor switch register |
| REASSIgn[1] (AC) | 21 | Assign device (SIXBIT name in AC+1) to job number in AC; skip if successful |
| TIMER[1] (AC) | 22 | Load AC with time of day in jiffies (clock ticks) |
| MSTIME[1] (AC) | 23 | Load AC with time of day in milliseconds |

Table 4 (Cont.)

| Name | n | Description |
|---|---|---|
| GETPPN[1] (AC) | 24 | Load $AC_L$ with proj number, $AC_R$ with prog number of job whose number is in AC |
| TRPSET[1] (AC) | 25 | Enter user I/O mode; if $AC_L$ = 40 to 57, put $C(C(AC)_R)$ properly relocated into $C(AC)_R$; skip if no error |
| TRPJEN[1] | 26 | Dismiss exec mode interrupt and restore PC from address in .+1 |
| RUNTIMe[1] (AC) | 27 | Load AC with accumulated run time (ms) of job whose number is in AC (0 = current job) |
| PJOB[1] (AC) | 30 | Load AC with job number of current job |
| SLEEP[1] (AC) | 31 | Delay running of job for C(AC) seconds. |
| | ⩾32 | Not used |

NOTES:

(AC) - AC used (D)- User's I/O channel number used (in AC field)

[1]Not available in 10/20 or 10/30 single-user Monitors

[2]10/50 Monitor only; available only during LOGIN procedure, not for user

[3]Feature under development; may vary

## DATE STORAGE

12-bit field $31\{12(year-1964)+(month-1)\}+(day-1)$

1 Jan 1964 to 4 Jan 1975

## FILE PROTECTION BITS

9-bit field

| PROT CHG PROT | READ PROT | WRITE PROT | PROT CHG PROT | READ PROT | WRITE PROT | PROT CHG PROT | READ PROT | WRITE PROT |
|---|---|---|---|---|---|---|---|---|
| OWNER | | | PROJECT | | | OTHERS | | |

## COMMAND LISTS

| | |
|---|---|
| -n,location-1 | Transfer n words starting at location |
| 0,address | Take next command from address |
| -n,0 | Skip n words of data (hardware channel only) |
| 0,0 | Stop |

Table 5

Cross Reference Listing of I/O Programmed Operator Symbols[1]


| | | | | | |
|---|---|---|---|---|---|
| xxxCHL | D33 | DIN | D29(t2),D30 | IODEND | D27(t1) |
| xxCHN | D33 | DINI | D29(t2) | IODERR | D27(t1) |
| xxxSAV | D33 | DLK | F23,D29(t2) | IODTER | D27(t1) |
| xxxXIT | D33 | DMT | D29(t2) | IOEND | D27(t1) |
| | | DOU | F18,D29(t2),D30 | IOFST | D27(t1) |
| ASSASG | F12 | DRL | F21,D29(t2),D30 | IOIMPM | D27(t1) |
| ASSCON | D28(t1) | DRN | D29(t2) | IONRCK | D27(t1) |
| ASSPRG | D28(t1) | DSI | D29(t2) | IOPAR | D27(t1) |
| | | DSO | D29(t2) | IOSETC | F15,F18 |
| BUFC1 | F13 | DVAVAL | D28(t1) | IOTEND | D27(t1) |
| BUFCLC | F13 | DVCDR | D27(t1) | IOW | D27(t1) |
| BUFCLR | F18 | DVDIR | D28(t1) | IOWC | D27(t1) |
| | | DVDIRIN | D27(t1) | | |
| CALIN | D6,F14,F15,F16 | DVDIS | D28(t1) | JBTADR | D1 |
| CHAN | D32 | DVDSK | D27(t1) | JBTSTS | D1 |
| CHn | D32 | DVDTA | D28(t1) | JOBFF | D5 |
| CLOSE1 | D9,F19,F21,F23 | DVIN | D28(t1) | JOBJDA | D1,D2(f1) |
| CLRDDB | F22 | DVLNG | D28(t1) | JOBHCU | D1 |
| | | DVLPT | D27(t1) | JOBPFI | D1 |
| DCL | F20,D29(t2), D30 | DVMTA | D28(t1) | | |
| | | DVOUT | D28(t1) | LOOKB | D2(f1) |
| DCLI | F19,D29(t2) | DVPTP | D28(t1) | | |
| DCLO | D29(t2),D30 | DVPTR | D28(t1) | NULL | D30 |
| DCLR | D29(t2) | DVTTY | D28(t1) | | |
| DDI | D5,F14,D29(t2) | | | OBUFB | D2(f1) |
| DDO | F17,D29(t2) | ENTRB | D2(f1) | OCLOSB | D2(f1) |
| DEN | F23,D29(t2) | | | OUT | D8,F17,F20 |
| DEVADR | D4(f2),D25, D26(f10) | IBUFB | D2(f1) | OUT2 | D8,F18 |
| | | ICLOSB | D2(f1) | OUTA | D9,F17,F18 |
| DEVBUF | D5,D25,D26(f10) | IN | D6,F14 | OUTBFB | D2(f1) |
| DEVCHR | D5,D24,D26(f10) | IN1 | D6,D7,F14 | OUTDMP | D8,F17 |
| DEVCTR | D4(f2),D25,D26 (f10) | INBFB | D2(f1) | OUTF | D9,F18 |
| | | INDMP | F14 | OUTF1 | D9,F18 |
| DEVDAT | D6,D24 | INEOF | D7,F15 | OUTPB | D2(f1) |
| DEVEXT | D25,D26(f10) | INEOFE | D8,F15 | OUTS | D9,F18 |
| DEVFIL | D25,D26(f10) | INITB | D2(f1) | | |
| DEVIAD | D5,D6,D25, D26(f10) | INPB | D2(f1) | PRJPRG | D1 |
| | | INPT0A | D6,D7,F15 | | |
| DEVIOS | D5,D24,D26(f10), D27(t1) | INPT0C | D7,F15 | RELEA0 | D10,F12,F21 |
| | | INPT1 | D7,F15 | RELEA1 | D10,F21 |
| DEVLOG | D25,D26(f10) | INPT2 | D6,D7,F15 | RELEA2 | D10,F21 |
| DEVMOD | D25,D26(f10) | INPUT2 | D6,F15 | RELEA3 | D10,F21 |
| DEVNAM | D24,D26(f10) | INPUT3 | F16 | RELEA4 | D11,F21 |
| DEVOAD | D5,D25,D26(f10) | INPUTF | D6,F16 | RELE4A | D11,F21 |
| DEVPTR | D4(f2),D25,D26 (f10) | IO | D27(t1) | RELEA5 | D11,F21 |
| | | IOACT | D27(t1) | RELEA6 | D11,F22 |
| DEVSER | D1,D24,D26(f10) | IOBEG | D27(t1) | RELEA7 | D11,F22 |
| DEVSRC | F12 | IOBKTL | D27(t1) | RELEA9 | D11,F22 |
| DGF | D29(t2) | IOBOT | D27(t1) | | |
| DHNG | D29(t2),D30 | IOCON | D27(t1) | SYSDEV | D2(f1) |


[1] A D preceding a page number indicates that a description of the item is found on that page; an F indicates that the item appears in a flow chart on that page. (tn) = Table (fn) = Figure

Table 5 (Cont.)

| | | | | | |
|---|---|---|---|---|---|
| TTYATC | D27(t1) | UCLSBI | D9,F19 | UINITC | F12 |
| TTYBIU | D27(t1) | UCLSBO | D10,F20 | UOBFI | F13 |
| TTYUSE | D27(t1) | UDEN | F23 | UOUT | D8,F17 |
| | | UDLK | F23 | UOUTBF | F13,F18 |
| UCLS0 | D10,F19 | UDLKC | F23 | USRJDA | D1,D2(f1) |
| UCLS1 | D10,F20 | UERROR | F15 | | |
| UCLS2 | D10,F20 | UINBF | F13,F16 | WAIT1 | F14,F17,F18, |
| UCLS2A | D10,F20 | UINIT | F12 | | F19,F20,F21, |
| UCLS2B | D10,F20 | UINITA | F12 | | F23 |
| UCLS3 | D10,F20 | UINITB | F12 | WSYNC | F14,F17,F18 |

DIGITAL EQUIPMENT CORPORATION ▫ MAYNARD, MASSACHUSETTS

Printed in U.S.A.